FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO

Ensinamento por demonstração com controlo de força e inspeção por visão em células de polimento robotizadas

Nuno Jorge Pais Martins Rodrigues

MESTRADO INTEGRADO EM ENGENHARIA ELETROTÉCNICA E DE COMPUTADORES

Orientador: António Paulo Gomes Mendes Moreira

Coorientador: Paulo José Cerqueira Gomes da Costa

Julho de 2016

Resumo

Nesta dissertação, estudou-se a viabilidade de um sistema de polimento robotizado programado por demonstração e desenvolveu-se um sistema de inspeção automática por visão artificial de peças com superfície altamente refletora e espelhada.

No que diz respeito ao primeiro, desenvolveu-se uma ferramenta de polimento onde encaixava um marcador luminoso e um sensor de força, para que as trajetórias e as forças demonstradas por um utilizador, pudessem ser registadas através do recurso a uma *framework* de transferência de *know-how* chamada *6DMimic*.

Estudou-se o controlo de força do manipulador industrial *UR5*, da *Universal Robots*, para ser utilizado no polimento robotizado. Para além da implementação dos protocolos de comunicação com o sensor de força e o *6DMimic*, desenvolveram-se algumas funções em *C++* para que fosse possível ler o estado interno e controlar o manipulador referido, a partir de um PC.

Foi desenvolvido um método de calibração, para transformar corretamente as trajetórias da zona de demonstração para a zona de atuação do *UR5*. Estas trajetórias foram suavizadas, através de um algoritmo implementado para diminuir o número de instruções de movimentação enviadas para o manipulador industrial.

Relativamente ao sistema de inspeção automática por visão artificial, desenvolveu-se uma técnica de condicionamento do ambiente de aquisição e um algoritmo de deteção de defeitos.

O condicionamento do ambiente de aquisição foi extremamente necessário devido à elevada refletividade das peças, o que provocava o aparecimento de reflexões especulares indesejáveis. Este foi feito através do isolamento das peças, relativamente ao ambiente exterior, e através da utilização de iluminação do tipo difusa frontal de campo escuro.

Foram comparadas várias abordagens para segmentar os defeitos que tipicamente apareciam nas peças. Os melhores resultados foram obtidos através da utilização do gradiente morfológico, no realce das orlas (defeitos) presentes nas imagens adquiridas. A identificação dos defeitos foi feita com base na imagem previamente segmentada, através de funções da biblioteca *OpenCV*.

Por último, integrou-se no sistema de inspeção por visão o mesmo manipulador industrial, para que fosse feito o varrimento de toda a superfície das peças.

Abstract

In this dissertation, was studied the viability of a robotic polishing system programmed by demonstration and was developed an automatic vision inspection system for highly reflective and mirrored surface pieces.

On what concerns the first topic, a polishing tool was developed, where a marker and a force sensor were added, in order to record the trajectories and the forces demonstrated by a user, using a framework for human-robot skill transfer called 6DMimic.

UR5's industrial robot force control, was tested to be used in robotic polishing. Were implemented the communication protocols with force sensor, 6Dmimic and UR5. For the latter, were developed some C++ functions, to make possible reading the internal state and control the industrial robot from a PC.

A calibration method was developed to properly transform trajectories from demonstration's zone to UR5's actuation zone. This trajectories were filtered by an algorithm to decrease the number of instructions sent to the industrial robot.

Concerning the automatic vision inspection system, was developed a technique for conditioning the image acquisition space and a defect detection algorithm.

The conditioning of image acquisition space was required due to the extremely high reflectivity pieces, which caused the appearance of unwanted specular reflections. This was done by isolating the pieces from the external space, and by the use of diffuse front lighting illumination (dark field).

Several approaches, about segmentation of piece's considered defects, were compared. Best results were obtained using morphological gradient in enhancement of edges (defects). The identification of defects was done using previously segmented image, through OpenCV functions.

Finally, was integrated the same industrial robot in the automatic vision inspection system, to be do the scanning of the entire piece's surface.

Agradecimentos

Em primeiro lugar, quero agradecer à minha família, particularmente aos meus pais, ao meu irmão e à minha avó, por todo o apoio que me deram. Sem eles, hoje não seria a pessoa que sou.

Depois, à minha Andreinha, por todo o carinho, boa disposição e paciência, ajudando-me sempre nos momentos em que as coisas não correram tão bem.

Aos intervenientes diretos na realização deste projeto:

- Em especial, ao meu orientador, Professor Doutor António Paulo Moreira, por me ter orientado neste projeto. A sua disponibilidade e prontidão para me ajudar nos problemas que surgiram, foram cruciais para que soubesse sempre qual era o caminho a seguir;
- Ao meu coorientador, Professor Doutor Paulo Costa, e ao Doutorado em Engenharia Andry Pinto, por me terem orientado e ajudado na elaboração do algoritmo de inspeção por visão artificial;
- Ao Doutorado em Engenharia Marcos Ferreira, por toda ajuda prestada com o sistema de demonstração de trajetórias – *6DMimic*;
- Ao Doutorado em Engenharia Luís Rocha, por todo o conhecimento partilhado sobre manipuladores industriais;
- Ao Sr. Fernando Guedes e ao Jorge Barbosa, por todo o apoio técnico prestado ao longo da realização deste projeto;
- Ao INESC TEC e à Lobo & Filhos LTD, pela disponibilização de todos os recursos necessários para a realização de testes.

Por último, mas não menos importante, quero agradecer a todos os meus colegas de curso, por toda a motivação e entreajuda ao longo dos últimos anos.

Nuno Jorge P. M. Rodrigues

"It always seems impossible
until it's done."

Nelson Mandela

Conteúdo

Lista de Figuras

Lista de Tabelas

Abreviaturas e Símbolos

API	*Application Programming Interface*
CAD	*Computer Aided Design*
CAM	*Computer Aided Manufacturing*
CCD	*Charge-coupled device*
FEUP	Faculdade de Engenharia da Universidade do Porto
FPS	*Frames* por segundo
FTP	*File Transfer Protocol*
IMU	*Inertial Measurement Unit*
INESC TEC	Instituto de Engenharia de Sistemas e Computadores - Tecnologia e Ciência
IP	*Internet Protocol*
LED	*Light Emitting Diode*
LTD	Limitada
NC	*Numerical Control*
OpenCV	*Open Source Computer Vision Library*
PID	Proporcional integral derivativo
PoE	*Power over Ethernet*
RAM	*Random Access Memory*
TCP	*Transmission Control Protocol*
UDP	*User Datagram Protocol*
USB	*Universal Serial Bus*
VDC	*Voltage Direct Current*
3D	Espaço tridimensional

A	Ampere
dB	Decibel
GB	Gigabyte
g	Grama
$^{\circ}$	Grau
$^{\circ}$C	Grau Celsius
Hz	Hertz
m	Metro
μ	Micro
mm	Milímetro
N	Newton
px	Pixel
kg	Quilograma
V	Volt
W	Watt

Capítulo 1

Introdução

Esta dissertação tinha como objetivo o estudo da viabilidade de um sistema de polimento robotizado programado por demonstração humana e o desenvolvimento de um sistema de inspeção automática por visão artificial de peças com superfície altamente refletora e espelhada.

Neste capítulo, é apresentado o problema em causa, a motivação que levou à sua resolução, os objetivos propostos e, por último, a estrutura deste documento.

1.1 Apresentação do problema

A utilização de manipuladores robóticos nas operações de polimento de pequenas peças, é ainda muito escassa devido à complexidade de programação envolvida. O tempo necessário para se desenvolver manualmente uma aplicação é muito elevado pois os movimentos são complexos, muito variados e é necessário o controlo de força. Este facto, coloca de fora os manipuladores no caso de séries médias e pequenas e com elevado grau de customização que, no mundo atual, são cada vez mais frequentes. Posto isto, pretendeu-se inovar no sentido de se desenvolverem novas técnicas de programação de manipuladores, baseadas na programação por demonstração e no registo das forças envolvidas.

Esta parte do projeto, realizou-se em colaboração com a empresa TALUS Robotics, S.A., que já possuía um sistema similar para pintura com uma patente da FEUP licenciada.

A inspeção de qualidade é um passo extremamente importante na produção industrial e a visão artificial é cada vez mais utilizada nesta área. Normalmente, esta inspeção é feita por indivíduos qualificados, no entanto, como depende do olho humano muitas vezes está sujeita a erros, como é o caso do efeito do cansaço. Testes mostraram que a inspeção por visão humana não é mais do que 80% eficaz [6].

A inspeção por visão de peças polidas ou pintadas com superfície altamente refletora e espelhada, é particularmente complicada devido ao aparecimento de reflexões especulares ao longo da superfícies. Estas reflexões provocam a saturação dos pixeis nas zonas afetadas, impossibilitando a recolha de qualquer informação nessas zonas da peça. Desta forma, nesta dissertação

pretendeu-se ainda desenvolver, para além de um algoritmo de visão artificial, técnicas especiais de iluminação que permitam a inspeção destas peças.

A empresa Lobo & Filhos LTD, acompanhou o projeto, pois tem bastante interesse nos resultados finais devido ao tipo de peças que produz.

1.2 Motivação

A automatização de processos é a minha principal área de interesse neste ramo de engenharia. Ter tido a oportunidade de estudar soluções nesta área com a possibilidade de realizar testes em ambiente industrial, foi sem dúvida uma motivação extra. Agora terminado este projeto, espero ter colmatado algumas das necessidades de inovação apresentadas.

Este projeto foi desenvolvido não só com a colaboração da TALUS Robotics, S.A. e da Lobo & Filhos LTD, como já foi referido, mas também com a colaboração do INESC TEC. Estas parcerias foram uma mais-valia, uma vez que contribuíram com vastos conhecimentos nestas áreas e estimularam o trabalho de equipa. A valorização profissional resultante da interação com as empresas referidas, foi outro aspeto bastante importante, sendo sempre um fator motivador ao longo da realização desta dissertação.

1.3 Objetivos

Os objetivos propostos para o estudo da viabilidade de um sistema de polimento robotizado programado por demonstração humana, consistiram no desenvolvimento de uma aplicação capaz de:

- Gravar numa base de dados as trajetórias e as forças envolvidas na demonstração humana da tarefa de polimento;
- Permitir ligeiros ajustes nos movimentos, quer nas trajetórias como nas velocidades, e nas forças envolvidas, para acelerar o processo;
- Gerar o código a descarregar para o manipulador industrial através dos pontos das trajetórias e das forças, adquiridos durante a demonstração;
- Disponibilizar uma interface com o utilizador para sinalizar o início e o fim das demonstrações.

Nesta parte do projeto, outro objetivo proposto foi averiguar se seria viável utilizar o controlo de força do manipulador industrial *UR5* da *Universal Robot*, sem ser necessário o uso de um sensor de força externo.

Relativamente ao desenvolvimento do sistema de inspeção automática por visão artificial de peças com superfície altamente refletora e espelhada, os objetivos propostos foram:

- Desenvolver uma técnica de iluminação robusta que permitisse a captação correta de imagens para serem analisadas em ambiente industrial;
- Elaborar um algoritmo de inspeção de peças com tempo de processamento pequeno, que fosse capaz de identificar defeitos e classificar o estado de polimento atual (no caso concreto das peças polidas);
- Comparar as abordagens baseadas em *software* grátis (licença BSD), com abordagens baseadas em *software* comercial (*HALCON*).

Por último, pretendia-se a integração dos sistema de polimento robotizado e de inspeção automática por visão, para que as peças ao fim de serem polidas, fossem imediatamente inspecionadas.

1.4 Estrutura do documento

Este documento está divido segundo os seguintes capítulos:

- **Capítulo 2 - Revisão Bibliográfica:** Estudo realizado sobre sistemas de manipulação robotizados com controlo de força, sobre o sistema de demonstração de trajetórias utilizado e sobre sistemas de visão aplicáveis à inspeção de peças;
- **Capítulo 3 - Ensinamento por demonstração com controlo de força:** Numa primeira fase, são apresentados todos os periféricos utilizados pelo sistema de polimento robotizado programado por demonstração humana, assim como a explicação da comunicação estabelecida com eles. De seguida, é explicada a calibração do sistema de demonstração de trajetórias e a forma como estas foram suavizadas. Por último, são apresentados os resultados obtidos através dos testes realizados ao controlo de força do manipulador industrial *UR5*;
- **Capítulo 4 - Inspeção por visão artificial de peças altamente refletoras e espelhadas:** Exposição de todo o sistema de inspeção automática por visão artificial desenvolvido. No inicio são apresentadas as peças e respetivos defeitos considerados. Posteriormente, é explicada a técnica de condicionamento do ambiente de aquisição adotada e o algoritmo de deteção de defeitos desenvolvido;
- **Capítulo 5 - Conclusões gerais e trabalho futuro:** Exposição das conclusões finais, do cumprimento dos objetivos estabelecidos e do trabalho futuro dos sistemas apresentados neste projeto;
- **Anexo A - Peças da ferramenta de polimento - Desenho detalhado:** Desenho detalhado das peças desenvolvidas no SOLIDWORKS para ferramenta de polimento;
- **Anexo B - Código C++ - Ensinamento por demonstração com controlo de força:** Apresentação das funções principais desenvolvidas para o sistema de polimento robotizado programado por demonstração;

- **Anexo C - Código C++ - Inspeção por visão artificial:** Apresentação das funções principais do algoritmo de deteção de defeitos desenvolvido;
- **Anexo D - Inspeção por visão artificial de peças altamente refletoras e espelhadas - Resultados finais:** Apresentação de todos os resultados obtidos nos testes realizados ao sistema de inspeção automática por visão artificial.

Capítulo 2

Revisão Bibliográfica

Este capítulo serve para apresentar a revisão bibliográfica da dissertação presente. Serão explicitados sistemas de manipulação robotizados com controlo de força, o sistema de demonstração de trajetórias a ser utilizado neste trabalho e sistemas de visão aplicáveis à inspeção de peças.

2.1 Sistemas de manipulação robotizados com controlo de força

Nesta secção, serão abordadas algumas estratégias de controlo de força que têm vindo a ser utilizadas em aplicações de polimento robotizadas.

A automação de processos de polimento é importante para melhorar a eficiência do acabamento da superfície das peças. Ultimamente, os manipuladores robóticos têm mostrado ser uma solução eficiente e económica para as aplicações de polimento automatizado [9].

O polimento é um processo mecânico baseado na remoção de uma camada muito fina do contorno da superfície. Uma operação idêntica ao polimento de peças é a rebarbação. A diferença entre os dois está na profundidade de material a remover, que é constante no polimento e não é na rebarbação (figura 2.1).

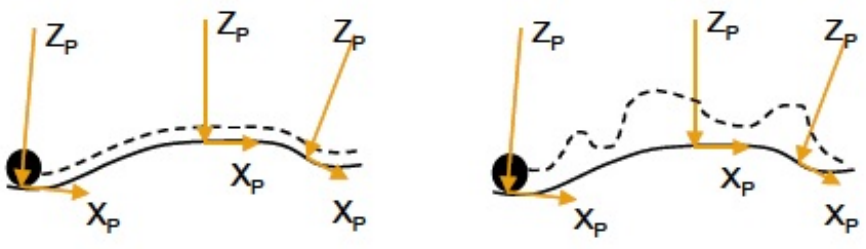


Figura 2.1: Processo de polimento (à esquerda) e processo de rebarbe (à direita) [1]

Preston [10] descreveu a remoção de material, dz, em qualquer ponto entre dois materiais em contato através de uma força F, movendo-se a uma velocidade relativa v_r e aplicada uma pressão p, através da seguinte equação

$$dz = \int k \cdot p \cdot v_r dt \tag{2.1}$$

onde o fator de Preston k, é uma constante de proporcionalidade que descreve a multiplicidade de influências tribo-químicas que são características para a configuração do processo em específico. No polimento, para se obter uma superfície lisa e precisa, deve-se manter constante a pressão exercida pela ferramenta sobre a peça. Desta forma, a equação 2.1 pode ser simplificada da seguinte forma

$$dz = k \cdot p \cdot \int v_r dt \tag{2.2}$$

Nas aplicações robotizadas de polimento, manipuladores controlados apenas através da posição estão extremamente dependentes da trajetória a percorrer e da complexa calibração exigida para se obter a profundidade de corte desejada. Foram comparados os métodos de controlo de posição e força de forma isolada. Verificou-se que com o controlo de posição, durante a trajetória a ferramenta pode deixar de estar contato com a peça, o que já não acontece com o controlo de força. As principais vantagens do uso do controlo de força são [1]:

- O desgaste da ferramenta é automaticamente compensado;
- A programação da trajetória a percorrer pelo robô é facilitada desde que o contato com a peça seja garantido pela malha do controlo de força.

Segundo Joaquim Silva em [2], a estratégia de controlo de força a adotar depende grandemente se estamos perante uma aplicação que necessita de **controlo ativo** ou **passivo**. No caso da aplicação a desenvolver necessitar de controlo de força para que a tarefa seja executada com sucesso, ou seja, sem se correr o risco de danificar as peças, estamos perante uma situação de **controlo de força passivo**. Neste tipo de controlo, apenas se pretende garantir que o valor da força de contato se mantém dentro de uma gama de valores pretendida. A tarefa de *pick and place*, desempenhada por manipuladores robóticos com sensores de toque nas garras, é um exemplo de uma aplicação com **controlo de força passivo**. Quando a eficiência e o grau de sucesso da tarefa está relacionado com as forças de contato, é necessário o **controlo de força ativo**. Neste tipo de controlo, pretende-se que a força de contato assuma valores mais precisos, consoante o tipo de tarefa a realizar. O polimento, rebarbagem, lixagem e montagem de componentes são exemplos de tarefas que necessitam deste tipo de controlo de força.

Várias estratégias de **controlo ativo** têm vindo a ser propostas, como por exemplo, métodos baseados em amortecimento ativo, métodos baseados em posição, controlo de impedância, controlo explícito da força, controlo híbrido, etc. No entanto, de todas as estratégias enunciadas, destacam-se o **controlo explícito da força**, o **controlo de impedância** e o **controlo híbrido**.

No **controlo explícito da força**, pretende-se levar a força de contato para um valor especificado, f_d, com o objetivo de minimizar a função de erro $e_f = f_d - f_{md}$, onde f_{md} é a força medida.

O controlo explícito pode ser **direto** ou **indireto**. No **direto**, incluem-se as estratégias que geram diretamente o sinal de atuação processando a função de erro anteriormente referida (figura 2.2). No **indireto**, é utilizado um anel exterior de força que fornece comandos de posição a um anel interior, constituído por um controlador de posição. Aqui, podem ser consideradas duas abordagens distintas:

- Os comandos de posição são calculados usando uma relação de admitância $A = Z^{-1} = (ms^2 + cs + k)^{-1}$ onde, m é a massa, c a constante de amortecimento e k a rigidez da superfície de contacto (figura 2.3);
- Os comandos de posição (normalmente sobre a forma de erro), são fornecidos por um controlador de força (figura 2.4).

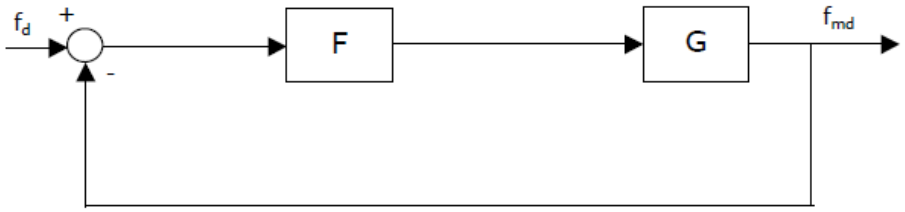


Figura 2.2: Controlo explícito direto (F representa o controlador de força e G a planta a controlar) [2]

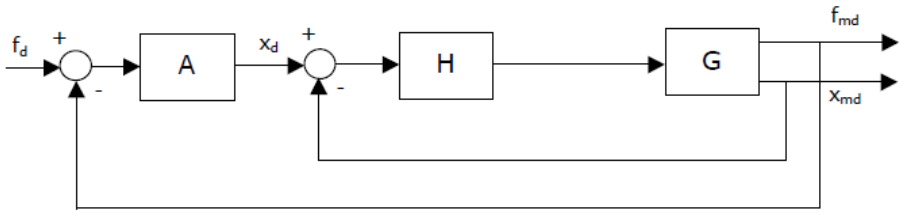


Figura 2.3: Controlo explícito indireto usando relação de admitância A (H é o controlador de posição, G é a planta a controlar, x_{md} é a posição medida e x_d é a posição desejada) [2]

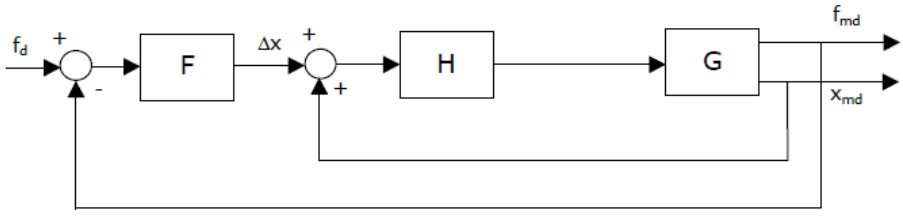


Figura 2.4: Controlo explícito indireto usando controlador de força F (H é o controlador de posição, G é a planta a controlar e Δx é o erro de posição) [2]

As estratégias de **controlo de impedância** conseguem-se através de uma relação dinâmica entre a posição do elemento terminal do manipulador industrial e a força que ele deve exercer.

Para isto, deve existir a realimentação da posição e da força. O diagrama da figura 2.5, ilustra este tipo de controlo, onde a relação de impedância é apresentada como vetor de comando f. Na relação de impedância de 1ª ordem, o vetor de comando é dado por

$$f = -c_d \dot{e}_p - k_p e_p \tag{2.3}$$

onde c_d é o amortecimento desejado e $\dot{e}_p = \dot{x}_d - \dot{x}_{md}$.

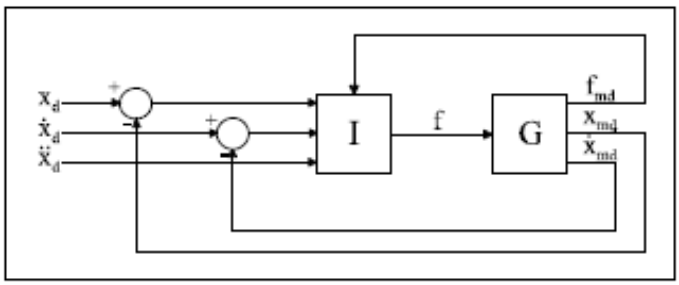


Figura 2.5: Controlo de impedância (I é o controlador de posição/força e G é a planta a controlar) [2]

O **controlo híbrido** consiste no controlo simultâneo da posição e da força (figura 2.6) [3].

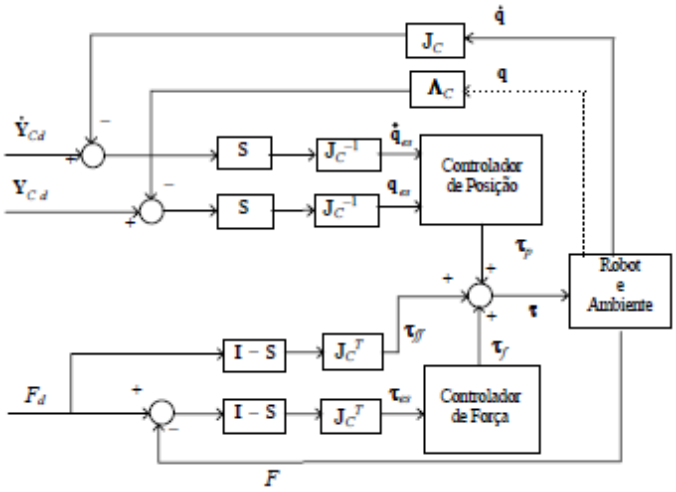


Figura 2.6: Controlo híbrido [3]

Seguidamente, são apresentadas soluções implementadas que utilizaram **controlo de impedância**:

- O sistema robotizado para lixar mobiliário com superfície de forma livre apresentado em [11], desenvolvido em 2007;
- No mesmo ano, em [12], Nagata, F. Hase, T., Haga, Z., Omoto, M., Watanabe, K. propuseram uma solução baseada em CAD/CAM com controlo de posição/força para polimento de

moldes com superfície curva. Esta solução foi executada com um controlo de força estável e de avanço exato ao longo da superfície. O controlo da força é feito com base nas forças de contato e de atrito cinético. A utilização de CAD/CAM permitiu realizar toda a operação de polimento sem ser necessário o ensinamento da trajetória e da direção de contato entre a ferramenta e o molde;

- Em 2009, desenvolveu-se uma solução em que foi utilizada a ferramenta de uma *Numerical Control machine (NC machine)* para fazer o acabamento da superfície de um molde. Contudo, a rigidez desta ferramenta apresentava não linearidades. Desta forma, desenvolveu-se uma rede neuronal para ajustar o amortecimento do sistema e não levar o sistema de controlo da força à instabilidade [13].

A seguinte solução utilizou **controlo explícito da força**:

- Muitas das vezes os manipuladores são programados em simuladores de forma *off-line*. Quando a ambiente de programação virtual não corresponde exatamente ao ambiente real onde irá atuar o manipulador podem surgir erros de posição consideráveis e as peças a serem trabalhadas podem ser danificadas. Mendes, N., Neto, P., Norberto Pires, J. e Paulo Moreira, A. em [14] (2010) propuseram uma solução para este problema instalando um sensor de força e torque no punho do manipulador controlado por um controlador do tipo *fuzzy-PI*.

Por último, seguintes soluções utilizaram **controlo híbrido**:

- Em 2000, uma aplicação industrial de rebarbação utilizando um controlador do tipo *fuzzy* adaptativo apresentada em [15];
- O sistema de polimento de lâminas de turbinas proposto em [1] (2008).

Hélder Miguel Tavares Moreira em 2010, propôs um sistema polimento de peças metálicas acopladas à extremidade de um manipulador. A força exercida pelo manipulador sobre a polidora é controlada com base no valor da velocidade da polidora, sem ter havido necessidade do uso de sensores de força [16].

Dada a necessidade do desenvolvimento de aplicações industriais robotizadas com controlo de força, ultimamente têm-se comercializado manipuladores já com o controlo de posição e força incorporado. O manipulador a ser utilizado nesta dissertação, o UR10 da *Universal Robots*, é um exemplo.

Foram desenvolvidas novas ferramentas de polimento, como em [9] (2015), estudou-se uma flange atuada por uma mola de ar controlada por um controlador do tipo *fuzzy-PID*. Através de simulações, chegou-se à conclusão que para a situação em causa um controlador *fuzzy-PID* é mais rápido, tem menos *overshoot* e tem melhor desempenho que um controlador PID tradicional. Concluiu-se ainda que a força de polimento é influenciada pela temperatura do gás, o volume da mola de ar, o coeficiente de elasticidade da mola, etc. Em 2009, a *Fraunhofer IPT* já tinha

desenvolvido uma cabeça de polimento orbital destinada ao polimento de superfícies com qualquer forma [10].

Uma das inovações nesta dissertação é o ensinamento por demonstração das forças envolvidas no processo de polimento e, podermos avançar para abordagens com forças não constantes ao longo da trajetória programada. A comunicação entre o robô e o operador é bastante importante uma vez que a experiência do último pode ser transferida para o processo automático. Em 2000, foi desenvolvida uma solução de polimento automatizado que utilizou o método de controlo de impedância com um compensador de posição/orientação controlado por um *joystick* [17]. Em 2001, Myoung Hwan, Choi e Woo Won, Lee desenvolveram um sensor de força/momento chamado COSMO-II para ensinamento intuitivo de robôs com 6 eixos [18]. Em [19] (2011), usou-se um sensor de força com três graus de liberdade para que o operador com um dedo, conseguisse ajustar a velocidade e força aplicada na peça durante o polimento feito por um robô do tipo ortogonal.

2.2 Sistema de ensinamento por demonstração

A operação de polimento, especialmente de superfícies não planas, requer movimentos complexos executados por operadores com anos de experiência. Nesta dissertação utilizar-se-á a *framework 6D-mimic*, que usa a tecnologia *sincrovision*, para registo dos movimentos que se pretendem executar com manipulador industrial. Nas secções seguintes, será apresentada a tecnologia *sincrovision* e a *framework 6D-mimic*.

2.2.1 Tecnologia *sincrovision*

A tecnologia *sincrovision* é uma invenção patenteada pela Faculdade de Engenharia da Universidade do Porto e licenciada à empresa FLUPOL, S.A. [20].

Trata-se de um sistema para captura de movimento que facilita a deteção de objetos no espaço recorrendo a visão artificial estereoscópica, utilizando emissores de luz visível de alta intensidade que se encontram sincronizados com o sistema de visão estereoscópico composto por pelo menos duas câmaras ligadas a um computador. Existe um módulo de sincronização que controla o momento de ativação dos emissores de luz, garantindo que todas as câmaras adquirem esse momento em simultâneo. O gráfico da figura 2.7 mostra os tempos de sincronização entre as câmaras e os emissores de luz (LEDs). A utilização de emissores de luz de alta intensidade sincronizados com as câmaras, faz com que estes necessitem de estar ligados por um breve período de tempo e, torna o sistema robusto visto que facilmente é distinguido o ambiente circundante. A intensidade aparente dos emissores é muito baixa, uma vez que estão ligados pouco tempo, pelo que não causa interferência aos utilizadores do sistema. O uso de emissores de várias cores do espetro visível permite uma fácil deteção pelo sistema de visão, evitando ambiguidades e reforçando a fiabilidade.

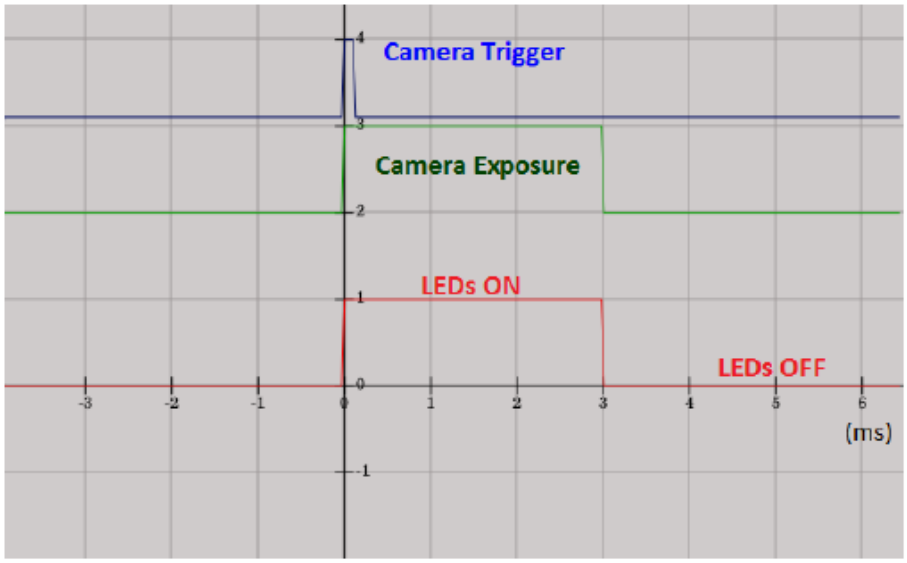


Figura 2.7: Tempos de sincronização entre as câmaras e os LEDs [4]

2.2.2 *Framework 6D-mimic*

A *framework 6D-mimic*, desenvolvida pelo Doutorado em Engenharia Marcos Ferreira no âmbito da sua tese de doutoramento [4], engloba um sistema de visão estereoscópica e um marcador luminoso (figura 2.8). Esta *framework* usa a tecnologia *sincrovision* apresentada na secção anterior, o que permitiu a leitura precisa e robusta do marcador luminoso mesmo em condições adversas, como é o caso de um ambiente industrial. São usados algoritmos otimizados para a forma do marcador e a cor dos LEDs, o que permite fazer uma reconstrução da trajetória do marcador com seis graus de liberdade – posição e orientação. A trajetória é ainda suavizada usando *B-Splines* em 3D e *splines* no espaço dos quaterniões. A *framework* foi implementada e testada com sucesso numa aplicação de pintura por pulverização na empresa Flupol.

A *6D-mimic* disponibiliza ainda um *software* com interface com o utilizador que permite a indicação do momento de início e fim dos movimentos a serem demonstradas. No final, é gerado um ficheiro de texto com as informações das trajetórias no espaço. O *software* por ser conectado através do protocolo UDP. Quando isto acontece, são enviados pacotes de informação nos instantes de início e fim do período de demonstração. Esta conexão serve como ponto de partida do trabalho a desenvolver nesta dissertação uma vez que, na demonstração do polimento de peças, têm de ser registados de forma sincronizada os movimentos e as forças envolvidas.

2.3 Sistemas de visão aplicáveis à inspeção de peças

A inspeção de peças polidas ou espelhadas é especialmente difícil devido às características altamente refletoras dessas peças, comportando-se como autênticos “espelhos”. Desta forma, a técnica de iluminação utilizada pode condicionar o ambiente de aquisição e facilitar a tarefa de

Figura 2.8: Marcador luminoso numa pistola de pintura [4]

inspeção. Nas secções posteriores, serão considerados alguns sistemas de polimento com inspeção por visão, algoritmos de inspeção normalmente usados com este tipo de peças e técnicas de iluminação especiais.

2.3.1 Sistemas de polimento com inspeção por visão

Têm vindo a ser desenvolvidos vários sistemas de visão aplicáveis a inspeção de peças polidas, como por exemplo:

- Em 2004, Chiu-Chun Ngan e Hon-Yuen Tam em [21], desenvolveram um sistema de visão composto por uma câmara CCD, um computador e um algoritmo de análise da imagem, para inspeção de moldes e matrizes polidas. O algoritmo de análise é baseado nas direções das fendas na superfície da peça a ser polida, o que dita o estado atual de polimento da peça. Foram obtidos resultados com rugosidades de superfícies de 0,02μm;

- Em 2010, foi desenvolvido um sistema complexo baseado em redes neuronais artificiais para polimento robotizado de superfícies. Em primeiro lugar, a superfície é aprendida pela rede neuronal que, posteriormente, dá instruções ao manipulador industrial para se mover. Os parâmetros de polimento desenvolvidos no sistema são o tempo e a força. Ao fim do ciclo de polimento, a peça é inspecionada pelo sistema de visão e, caso não esteja com as caraterísticas pretendidas, é novamente polida com os parâmetros tempo e força ajustados pela rede neuronal. A principal vantagem do uso de redes neuronais está no facto do sistema possuir aprendizagem e ter capacidade para se adaptar aos diversos tipos de defeitos que possam aparecer [22].

2.3.2 Algoritmos de visão para inspeção de peças

Para se obter um bom desempenho, o sistema de inspeção tem de ser robusto a problemas como a presença de ruído nas imagens, reflexão especular não uniforme ao longo da superfície, vários tipos de defeitos, etc. Quando se tem como objetivo a instalação deste tipo de sistemas em linhas de produção, como o caso de estudo presente, é necessário ter-se tempos de processamento pequenos para não atrasar toda a produção. Portanto, na elaboração do algoritmo de inspeção, deve conseguir-se estabelecer um ponto de equilíbrio entre a eficiência da inspeção e o tempo que ela demora a ser feita [23].

Vários algoritmos têm vindo a ser estudados, como por exemplo através do uso de *support vector machines* [24], através de preservação de bordas, duplo *threshold* e *threshold* adaptativo [23] e por comparação com uma imagem de referência [6]. No último, a imagem de deteção D pode ser definida como

$$D(i,j) = \begin{cases} 1, & \text{se } |I(i,j) - R(i,j)| > p(i.j) \\ 0, & \text{se } |I(i,j) - R(i,j)| \leq p(i.j) \end{cases} \tag{2.4}$$

onde I é a imagem capturada, R é a imagem de referência e p é o *threshold*.

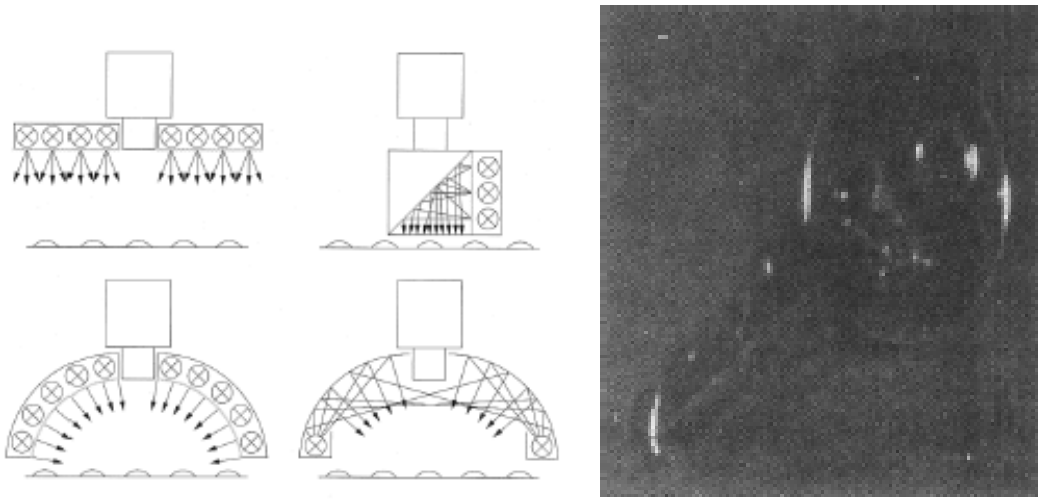
2.3.3 Técnicas de iluminação

O ambiente de aquisição de imagem pode e deve ser condicionado através da iluminação, permitindo desta forma realçar as características da imagem que se querem analisar. As técnicas de iluminação são dependentes de vários fatores, como a direção da luz, a posição da fonte de luz relativamente ao objeto e à câmara e, da quantidade de luz incidente no objeto e captada pela câmara [5].

Quanto à **direção da luz**, esta pode ser difusa ou direta. O uso de iluminação difusa dá origem a reflexões homogéneas, facilitando a eliminação de sombras e a redução de reflexões especulares (figura 2.9). A iluminação direta permite evidenciar detalhes da superfície dos objetos (figura 2.10). As peças metálicas após polidas têm caraterísticas a altamente refletoras, pelo que a incidência de luz provocará a reflexões especulares não uniformes devido à forma não plana da superfície. Desta forma, o uso de iluminação difusa poderá ser vantajoso.

Relativamente à **posição da fonte de luz relativamente ao objeto e à câmara**, pode usar-se iluminação frontal ou em contraluz. A iluminação em contraluz normalmente é usada quando se pretende evidenciar o contorno dos objetos, logo, como no caso de estudo não se pretende fazer isto, não será vantajoso o uso desta técnica (figura 2.11).

No que diz respeito à **quantidade de luz incidente no objeto e captada pela câmara**, poderá usar-se iluminação de campo claro ou escuro. A iluminação de campo escuro carateriza-se pela baixa quantidade de luz incidente na câmara e, normalmente é usada quando se pretende realçar



(a) Sistemas de iluminação (b) Imagem captada

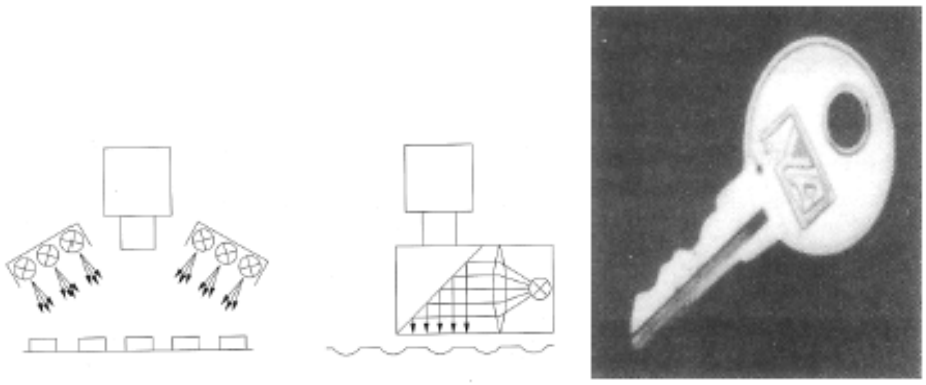
Figura 2.9: Iluminação difusa [5]

saliências e concavidades nos objetos (figura 2.12). A iluminação de campo claro carateriza-se pela incidência de uma quantidade razoável de luz na câmara.

A reflexão especular, referida anteriormente, é um problema inerente na inspeção deste tipo de superfícies uma vez que, quando é muito intensa, existe perda de informação sobre as zonas de reflexão uma vez que os *pixels* saturam em tons de branco (figura 2.13). Posto este problema, têm vindo a ser desenvolvidas técnicas de iluminação especiais para contornar este problema, tais como:

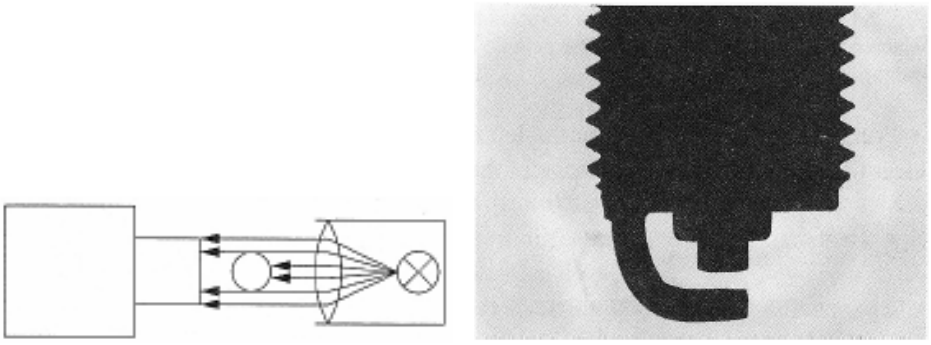
- Iluminação uniforme do cenário com lâmpadas florescentes e instalação de uma caixa de plástico 70% transparente na zona onde a peça é inspecionada. Este método não se tornou completamente eficaz uma vez que as zonas mais escuras da imagem não são uniformizadas e, dado que no caso de estudo em causa a comparação das imagens é feita através do nível de cinzento, defeitos nestas zonas podem não ser detetados (figura 2.14) [6]. Desta forma, seria interessante testar outros algoritmos de inspeção para casos como este;

- A desvantagem anterior é ultrapassada com o dispositivo criado em [7]. Trata-se de um anel de LEDs numa cúpula difusa. A luz difusa multi-direcional é refletida por quase todo o hemisfério, produzindo uma luz uniforme no centro da cúpula, onde as peças são bem iluminadas (figura 2.15).



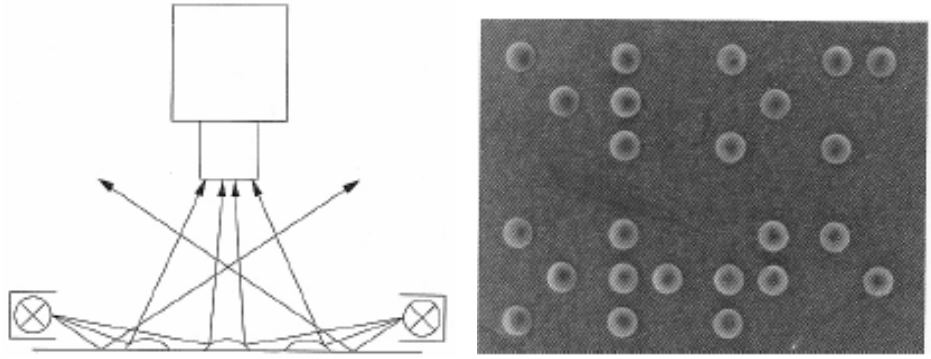
(a) Sistemas de iluminação (b) Imagem captada

Figura 2.10: Iluminação direta [5]



(a) Sistema de iluminação (b) Imagem captada

Figura 2.11: Iluminação em contraluz [5]



(a) Sistema de iluminação (b) Imagem captada

Figura 2.12: Iluminação de campo escuro [5]

Figura 2.13: Esfera metálica com reflexão especular [6]

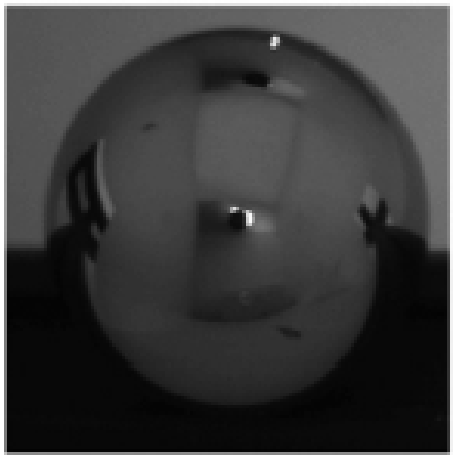
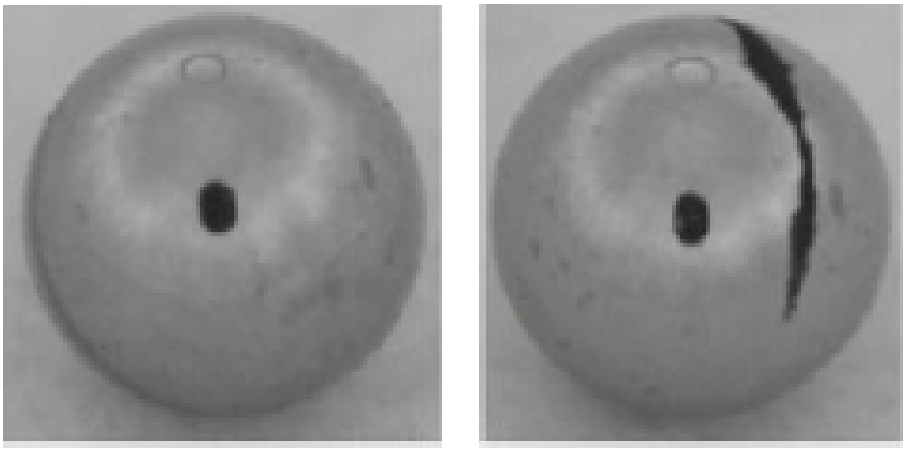


Figura 2.14: Esfera metálica com zonas escuras [6]



(a) Sem defeito (b) Com deifeito

Figura 2.15: Esfera iluminada uniformemente [7]

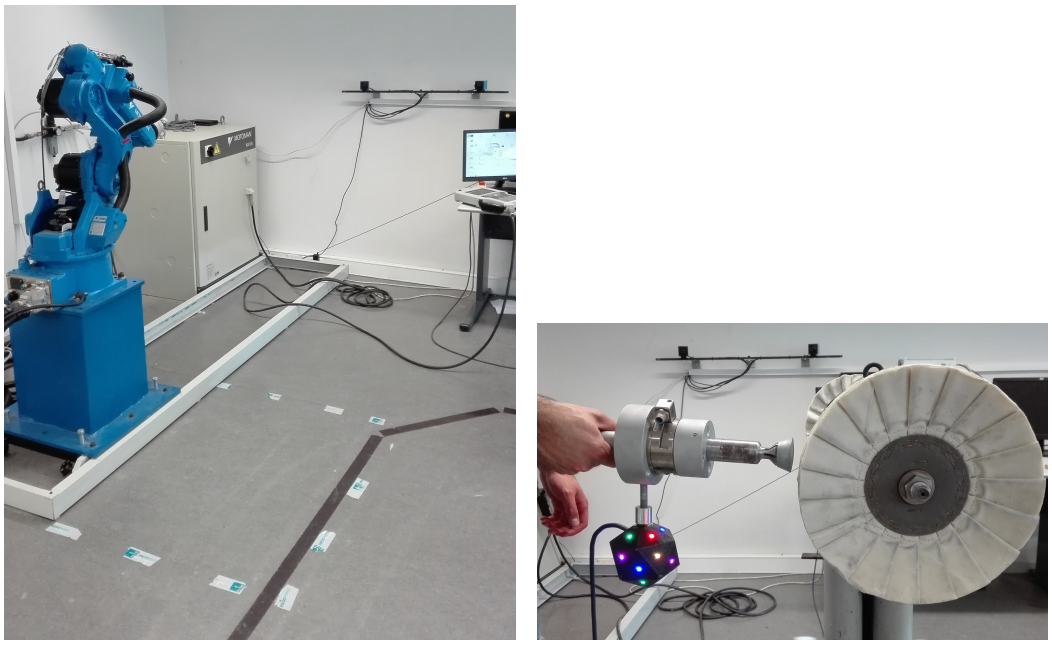
Capítulo 3

Ensinamento por demonstração com controlo de força

3.1 Visão geral da solução proposta

A solução proposta de polimento robotizado programado por demonstração humana consiste, numa primeira fase, na demonstração das trajetórias e das forças envolvidas nesta tarefa.

A demonstração das trajetórias foi feita com recurso a uma *framework* de transferência de *know-how* chamada *6DMimic*. Esta *framework*, através da tecnologia *sincrovision*, de um marcador luminoso e de um software dedicado, é capaz de gravar trajetórias demonstradas pelo operador. A gravação das forças é feita de forma sincronizada com as trajetórias. Na figura 3.1, é apresentada a zona de demonstração destas trajetórias e forças, assim como a polidora e a ferramenta de polimento utilizada.



(a) Zona de demonstração (b) Ferramenta de demonstração do polimento

Figura 3.1: Zona e ferramenta utilizadas na demonstração da tarefa de polimento por um operador

A partir das trajetórias e das forças demonstradas, seria gerado o código para o manipulador industrial *UR5*, da *Universal Robots*, para que este executasse a tarefa de polimento robotizado (figura 3.2). Neste projeto, apenas foi gerado o código a enviar para o manipulador relacionado com as trajetórias. O controlo de força foi somente testado e, devido à falta de tempo, não se conseguiu integrar no sistema final.

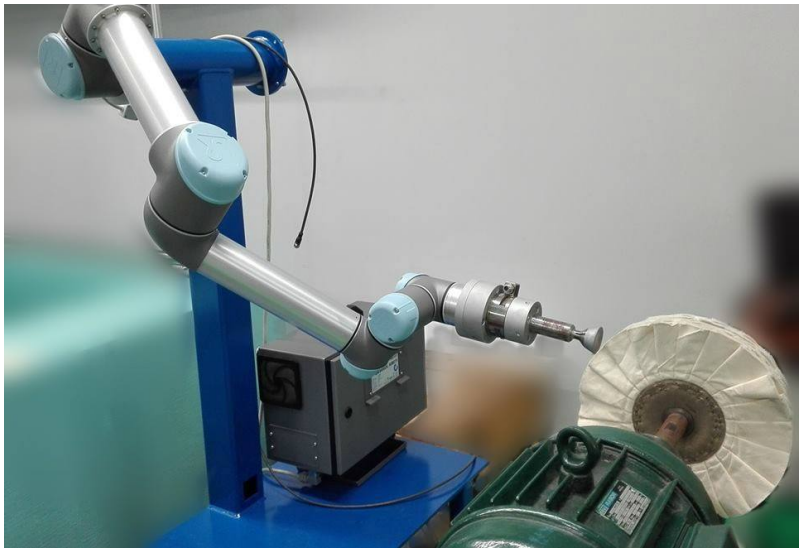


Figura 3.2: Manipulador industrial *UR5*, da *Universal Robots*, na execução da tarefa de polimento robotizado

3.2 *Hardware*

Nesta secção, será apresentado todo o *hardware* utilizado na solução proposta de polimento robotizado programado por demonstração.

3.2.1 Manipulador industrial – *UR5*

O manipulador industrial utilizado foi o *UR5* da *Universal Robots* (figura 3.3). Optou-se por este manipulador, visto que já possuía o controlo de força necessário para a tarefa de polimento robotizado e ser apropriado para funcionar junto de operadores humanos, sem necessidade de segurança extra. No entanto, era sabido à partida que este controlo não seria muito preciso, uma vez que é feito com base na leitura das correntes elétricas nos motores das articulações. Assim sendo, um dos objetivos desta dissertação, foi efetuar testes ao controlo referido, como será visto posteriormente.

Figura 3.3: Manipulador industrial *UR5* da *Universal Robots*

Na tabela seguinte, são apresentadas as especificações principais deste manipulador industrial. Especificações sobre as forças não são dadas pela marca.

Nº de articulações	6
Raio de trabalho	850 mm
Peso	18.4 kg
Carga de trabalho máxima	5 kg
Alcance das articulações	+/- 360°
Velocidades	Todas as articulações: 180°/s Ferramenta: Típico 1 m/s
Repetibilidade	+/- 0.1 mm
Comunicação	TCP/IP 100 Mbit: IEEE 802.3u, 100BASE-TX Ethernet socket & Modbus TCP
Classificação IP	IP54
Consumo de potência	Aproximadamente 200 watts num programa típico
Temperatura de funcionamento	0 - 50°C
Alimentação	100-240 VAC, 50-60 Hz
Link para a folha de especificações	Clique aqui
Link para o manual de utilizador	Clique aqui
Link para o manual de programação (via URScript)	Clique aqui

Tabela 3.1: Manipulador industrial *UR5* da *Universal Robots* - Especificações principais

3.2.2 Sensor de força/binário

O sensor de força/binário utilizado para gravar as forças demonstradas na tarefa de polimento, foi o *Gamma* da *ATI Industrial Automation* (figura 3.4). Este sensor, também foi usado para medir as forças controladas pelo manipulador industrial, como será visto mais à frente.

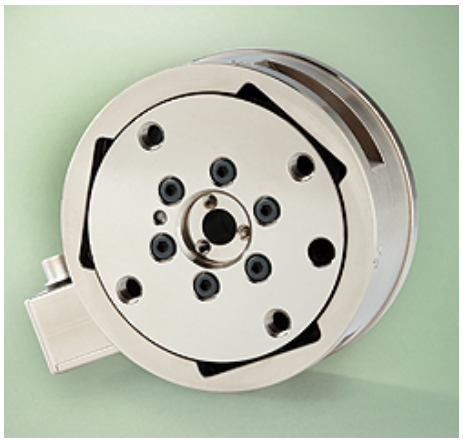


Figura 3.4: Sensor de força/binário *Gamma* da *ATI Industrial Automation*

Na tabela seguinte, são apresentadas as especificações principais deste sensor, para a montagem através de uma *Net Box* (figura 3.5), que será, também, apresentada de seguida. A calibração

utilizada foi a SI-65-5, dadas as forças envolvidas no polimento das peças. Este sensor era configurável através de uma página web disponibilizada pela marca.

Eixos	**Calibração**		
	SI-32-2.5	**SI-65-5**	**SI-130-10**
Alcances de medição			
Fx, Fy (N)	32	65	130
Fz (N)	100	200	400
Tx, Ty (Nm)	2.5	5	10
Tz (Nm)	2.5	5	10
Resolução			
Fx, Fy (N)	1/160	1/80	1/40
Fz (N)	1/80	1/40	1/20
Tx, Ty (Nm)	1/2000	10/13333	1/800
Tz (Nm)	1/2000	10/13333	1/800
Peso	0.255 kg		
Diametro	75.4 mm		
Altura	33.3 mm		
Link para a folha de especifcações	Clique aqui		

Tabela 3.2: Sensor de força/binário *Gamma* da *ATI Industrial Automation* - Especificações principais

O dispositivo *Net Box*, uma vez ligado ao sensor, serve para o alimentar e medir as suas componentes da força e do binário. Fornece interfaces de comunicação Ethernet e CAN bus, sendo compatível com os protocolos Ethernet padrão, Ethernet/IP^{TM} e DeviceNet^{TM} (todas as informações sobre este dispositivo podem ser consultadas aqui). A interface de comunicação utilizada foi a Ethernet, tendo sido alimentado pela mesma interface através de um *switch PoE* (*Power over Ethernet*).

Usou-se um computador pessoal com as caraterísticas apresentadas na tabela seguinte, para comunicar com a *Net Box*.

Figura 3.5: *Net Box*

Modelo	Samsung RC530-S04PT
Processador	Intel® Core™ i7-2670QM Quad Core (Sandy Bridge)
Memória RAM	6 GB
Disco Rígido	640 GB
Placa Gráfica	Sistema Híbrido NVIDIA® Optimus: GeForce™ GT 540M + Intel® HD Graphics
Sistema operativo	Windows 10 Pro, 64 bits

Tabela 3.3: Samsung RC530-S04PT - Especificações principais

A figura seguinte, apresenta as ligações efetuadas para adquirir os valores das forças e binários do sensor. A ligação entre a *Net Box* e o sensor foi feita com um cabo DeviceNet M12, e a ligação entre a *Net Box* e o *switch PoE* foi feita através de um cabo ethernet e um adaptador ethernet/DeviceNet M12.

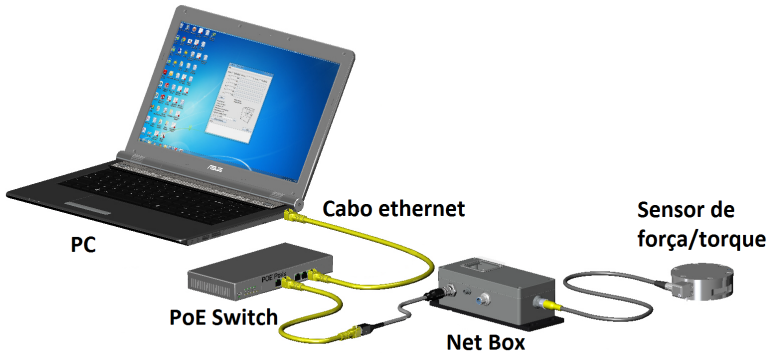


Figura 3.6: Esquema das ligações para comunicar com o sensor de força/binário

3.2.3 *Sincrovision* e marcador luminoso

Foi utilizada a *framework 6DMimic*, que utiliza a tecnologia *sincrovision*, na gravação das trajetórias demonstradas. O sistema de ensinamento por demonstração, já foi previamente apresentado na secção 2.2.

O último encontrava-se instalado com duas câmaras ligadas a um computador pessoal, onde estava instalado o *software 6DMimic* (figura 3.7). A demonstração das trajetórias foi feita com recurso a um marcador luminoso, dentro da área de demonstração captada pelas câmaras (figuras 3.8 e 3.9). A gravação é iniciada após o envio do comando "Start" para a *framework*, e terminada com o comando "Stop". Existia ainda um módulo de sincronização, que garantia o sincronismo das câmaras com os LEDs do marcador luminoso, onde era possível ligar e desligar todo o sistema (figura 3.10).

Figura 3.7: Câmaras do sistema de demonstração de trajetórias

(a) Apagado (b) Acesso

Figura 3.8: Marcador luminoso

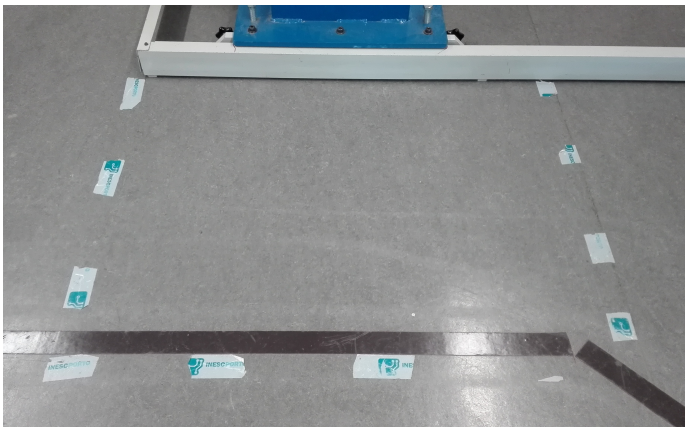


Figura 3.9: Área de demonstração captada pelas câmaras

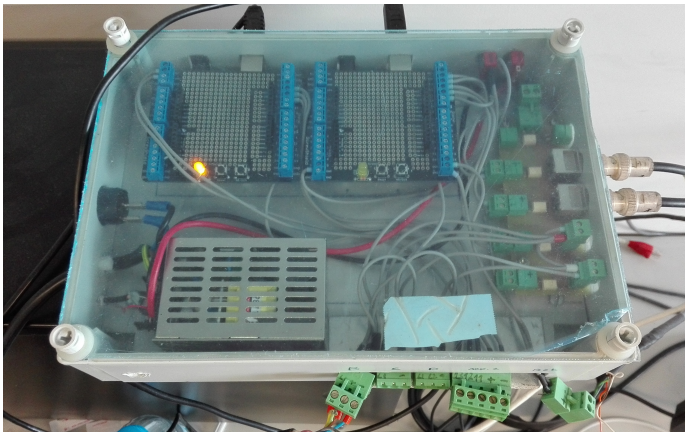


Figura 3.10: Módulo de sincronização das câmaras com os LEDs

3.2.4 Ferramenta de polimento

Desenvolveu-se no SOLIDWORKS uma ferramenta de polimento robotizado, possível de ser montada de várias formas, onde se tinha de inserir uma pega de polimento manual fornecida pela Lobo & Filhos LTD (onde encaixavam as peças a serem polidas (3.11)). Dependendo da montagem seguida, também seriam encaixados o marcador luminoso e o sensor de força, referidos nas secções anteriores.

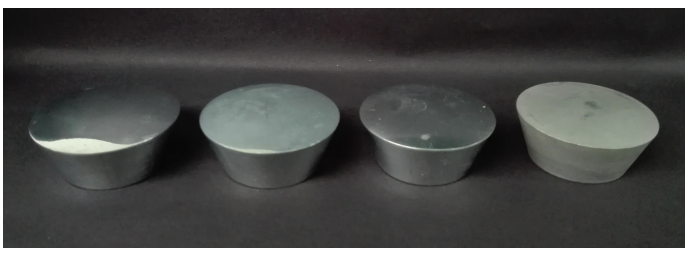


Figura 3.11: Peças com vários graus de polimento

A pega de polimento manual, foi adaptada de forma a que se conseguisse apertar e desapertar as peças encaixadas, através de uma simples rotação. A figura seguinte, mostra a referida pega, onde é possível observar um corte sobre o qual se deve rodar para apertar e desapertar a peça a polir ou polida.

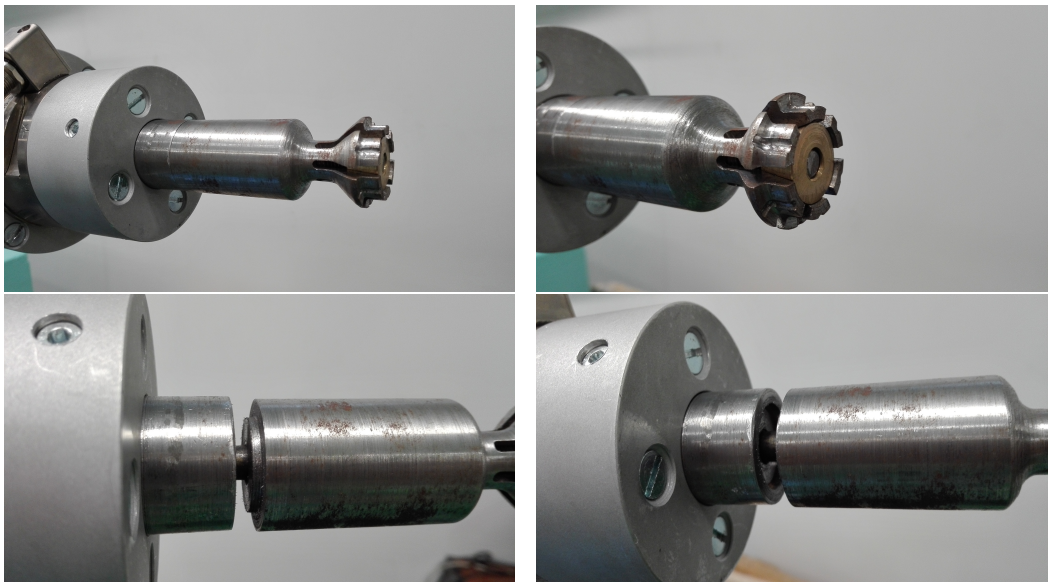


Figura 3.12: Pega de polimento manual adaptada

A ferramenta foi desenvolvida por peças, de forma a que permitisse as seguintes montagens distintas:

- Montagem 1 - Ferramenta de polimento robotizado, utilizada pelo manipulador industrial *UR5*, sem sensor de força/binário - Figura 3.13;
- Montagem 2 - Ferramenta de demonstração da tarefa de polimento - Figura 3.14;
- Montagem 3 - Ferramenta de polimento robotizado, utilizada pelo manipulador industrial *UR5*, com sensor de força/binário - Figura 3.15.

O propósito da montagem 1 era a operação normal de polimento com o *UR5*, sem o sensor de força, uma vez que este manipulador usava as correntes elétricas nos motores das articulações, para controlar a força na ferramenta. A montagem 2, serviu para o operador demonstrar a tarefa de polimento, sendo para isto necessário o sensor de força, o marcador luminoso e uma nova pega. A montagem 3, serviu para verificar se as forças demonstradas pelo operador eram corretamente controladas pelo *UR5*, através da comparação entre as forças demonstradas com as executadas pelo manipulador.

Posto isto, projetaram-se 5 peças, cujos desenhos detalhados estão presentes nas seguintes secções dos anexos:

- Peça 1 - Encaixe da pega de polimento manual - A.1;

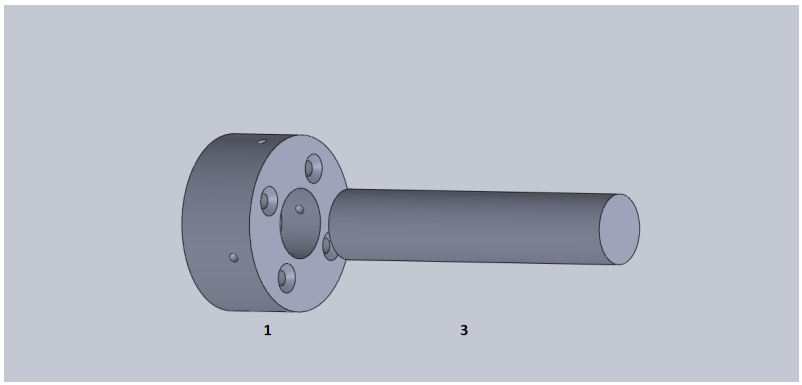
- Peça 4 - Pega da ferramenta de demonstração - A.2;

- Peça 5 - Encaixe da pega da ferramenta de demonstração, marcador luminoso e sensor de força - A.3;

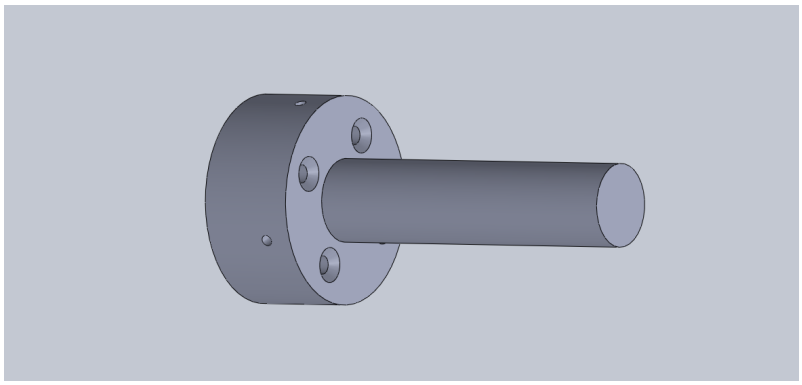
- Peça 6 - Adaptador para encaixar o sensor de força na peça 1 - A.4;

- Peça 7 - Adaptador para encaixar a peça 5 no *UR5* - A.5;

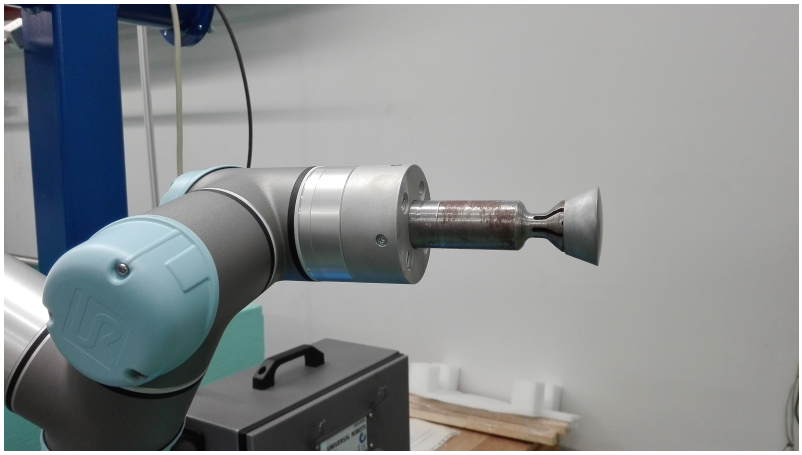
Tendo resultado daqui as montagens das figuras seguintes, onde a peça 3 corresponde à pega de polimento manual adaptada.



(a) Montagem no SOLIDWORKS - Vista expandida

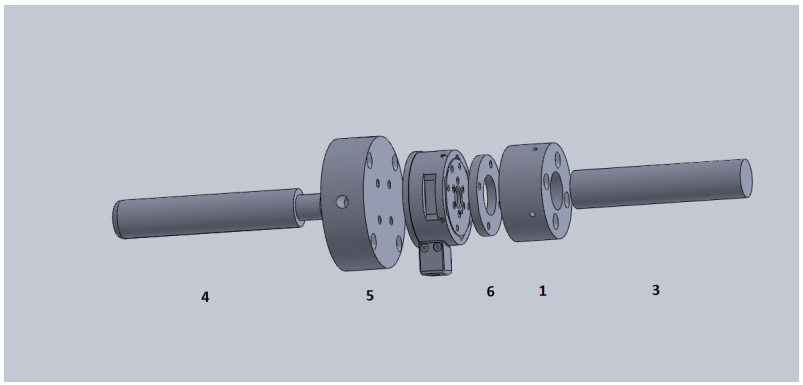


(b) Montagem no SOLIDWORKS

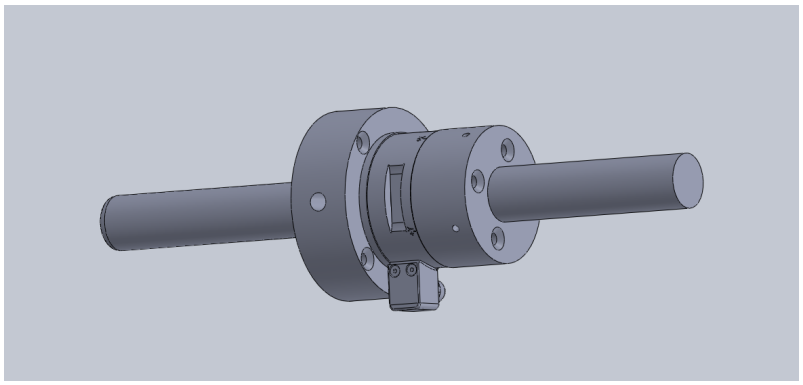


(c) Montagem real

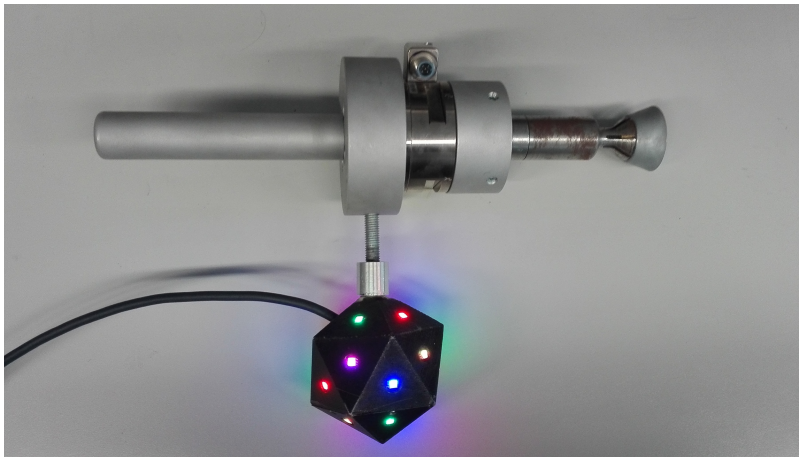
Figura 3.13: Montagem 1 - Ferramenta de polimento robotizado, utilizada pelo manipulador industrial, sem sensor de força/binário



(a) Montagem no SOLIDWORKS - Vista expandida

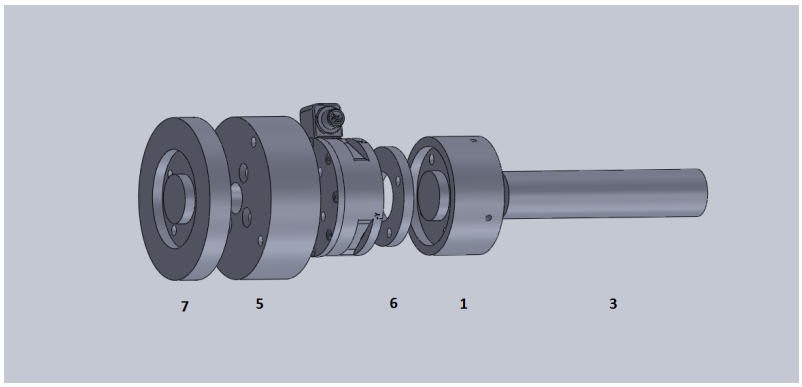


(b) Montagem no SOLIDWORKS

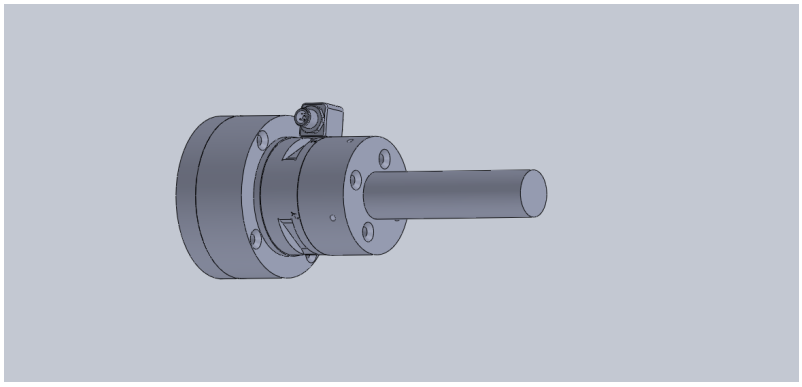


(c) Montagem real

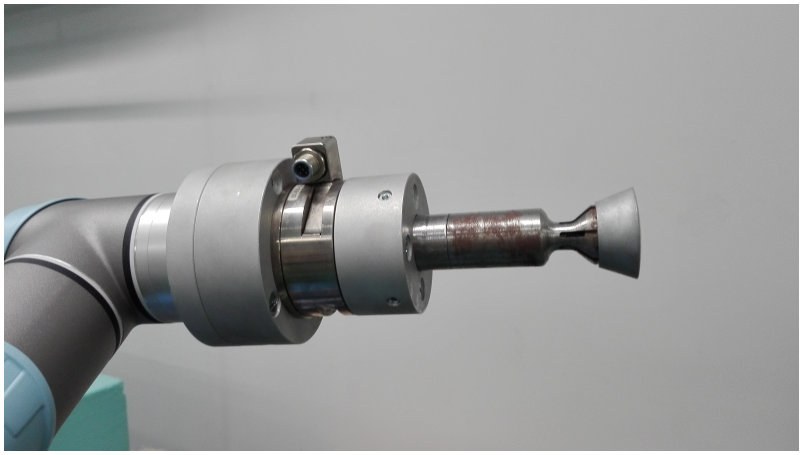
Figura 3.14: Montagem 2 - Ferramenta de demonstração da tarefa de polimento



(a) Montagem no SOLIDWORKS - Vista expandida



(b) Montagem no SOLIDWORKS



(c) Montagem real

Figura 3.15: Montagem 3 - Ferramenta de polimento robotizado, utilizada pelo manipulador industrial, com sensor de força/binário

3.3 Comunicação

Nesta secção, será explicada como foi estabelecida a comunicação entre o computador pessoal apresentado anteriormente, com o manipulador industrial *UR5*, o sensor de força/binário e o *6DMimic*.

3.3.1 *UR5*

Este manipulador, poderia ser programado a três diferentes níveis:

- Através do *teach pendant*;
- Através do envio de instruções *URScript* a partir de um PC;
- Através de uma API em C.

Escolheu-se a segunda alternativa, devido à documentação disponibilizada e à possibilidade de o fazer a partir de uma aplicação cliente a correr num PC, ligado ao controlador do manipulador usando *sockets* TCP/IP. Esta conexão foi feita através de um cabo ethernet, usando a porta 30002 e o IP estático do *UR5*, configurado através do *teach pendant*.

A leitura do estado do manipulador (p. ex. posição do ponto central da ferramenta, posições e velocidades das articulações e correntes elétricas nos motores), foi feita através da mesma comunicação especificada acima.

De seguida, será explicado com maior detalhe as instruções utilizadas para controlar este manipulador, assim como a descodificação da trama enviada com o seu estado interno.

3.3.1.1 Envio de instruções *URScript*

URScript é uma linguagem de programação, utilizada para programar o manipulador industrial em causa. Como qualquer outra linguagem de programação, esta tem variáveis, tipos, instruções de controlo de fluxo do programa, funções, etc. Toda a documentação sobre esta linguagem de programação pode ser consultada aqui.

A frequência da comunicação na porta utilizada era 10 Hz (10 instruções por segundo), como tal, o envio de cada instrução seguiu-se de uma espera de 100 ms, para se ter a certeza que as instruções eram recebidas pelo controlador.

Para a execução de trajetórias, foram enviadas instruções de configuração e longas listas de instruções de movimentação do ponto central da ferramenta. As instruções de movimentação, foram dadas para as poses da ferramenta. Estas poses, correspondem a uma posição (X, Y e Z) e a uma orientação (Rx, Ry e Rz) da ferramenta do manipulador em relação ao seu referencial base - $\begin{bmatrix} X & Y & Z & Rx & Ry & Rz \end{bmatrix}$. Os valores da posição foram usados em metros e os da orientação em radianos, segundo a notação de representação de rotações Eixo-Ângulo. Estas listas de movimentações, foram enviadas a partir dos pontos das trajetórias gravadas.

Todas as instruções foram enviadas sobre a forma de programas completos, que corriam no controlador do manipulador logo após o seu envio. No entanto, estas instruções também poderiam ser enviadas de forma isolada, sem estarem dentro de programas, e também corriam logo após ser concluído o envio de cada uma. Contudo, esta não seria uma forma cómoda de controlar o manipulador, uma vez que o envio de cada movimentação teria de ser precedido de um teste para averiguar se a movimento anterior já estaria concluído. Caso isto não fosse feito, a última movimentação enviada sobrepunha-se a todas as outras enviadas anteriormente. Com o envio de

programas completos, este problema não acontece, uma vez que cada movimentação só é executada após a conclusão da sua precedente.

A instrução de movimentação escolhida foi o *movel* (movimento linear no espaço da ferramenta), para que o movimento sobre o disco da polidora fosse linear. Para além da pose, esta função recebia uma aceleração, a, em m/s^2, uma velocidade, v, em m/s, um tempo, t, em s e um raio de tolerância para a paragem do ponto central da ferramenta, r, em m. O parâmetro t, sendo opcional, não foi usado uma vez que este tem prioridade sobre os parâmetros aceleração e velocidade. A aceleração e o raio de tolerância utilizados foram $4m/s^2$ e $0.012m$, respetivamente, uma vez que foi para estes valores que se obteve uma maior semelhança entre a trajetória demonstrada e a executada pelo manipulador. A velocidade da movimentação era retornada pelo *6DMimic*, como será visto mais à frente. Resumindo, a instrução de movimentação enviada tinha sempre o seguinte formato, onde a pose enviada tinha o formato indicado em cima (com os parêntesis retos):

- *movel(pose, a=4.0, v=v_ponto, r=0.012)*

Para além das instruções de movimentação, foram também enviadas nos cabeçalhos dos programas, as seguintes instruções de configuração do manipulador:

- *set_gravity* - Para especificar a montagem do manipulador. Esta especificação foi feita através de um vetor com magnitude e sentido inverso ao da aceleração da gravidade, expresso no referencial base do manipulador. Dada a montagem e o referencial base do *UR5* apresentados na figura 3.16, o vetor especificado foi $\begin{bmatrix} 9.82 & 0 & 0 \end{bmatrix}$;

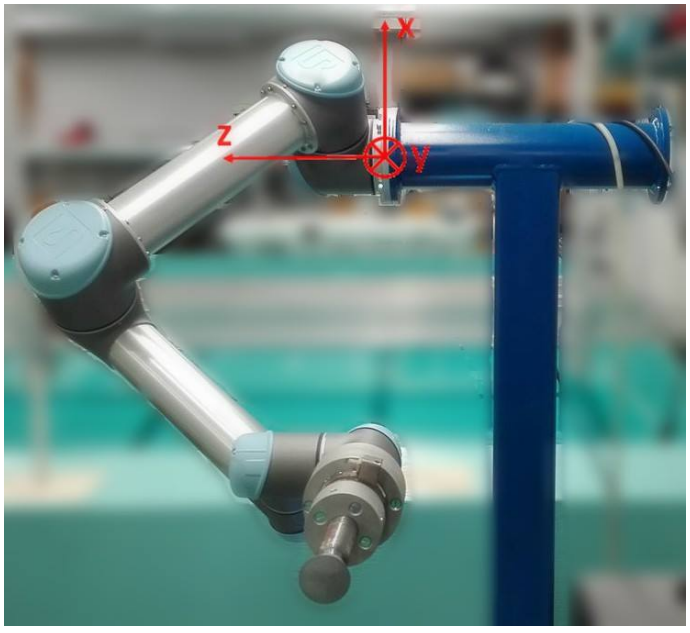


Figura 3.16: Montagem do *UR5*, numa estrutura *dual-arm*, e respetivo referencial base

- *set_tcp* - Para especificar o ponto central da ferramenta, expresso no referencial base do elemento terminal do manipulador (figura 3.17), sobre o formato de pose. Para a montagem

1 da ferramenta de polimento, a pose especificada foi $\begin{bmatrix}0 & 0 & 0.138 & 0 & 0 & 0\end{bmatrix}$ e para a montagem 3, $\begin{bmatrix}0 & 0 & 0.221 & 0 & 0 & 0\end{bmatrix}$;

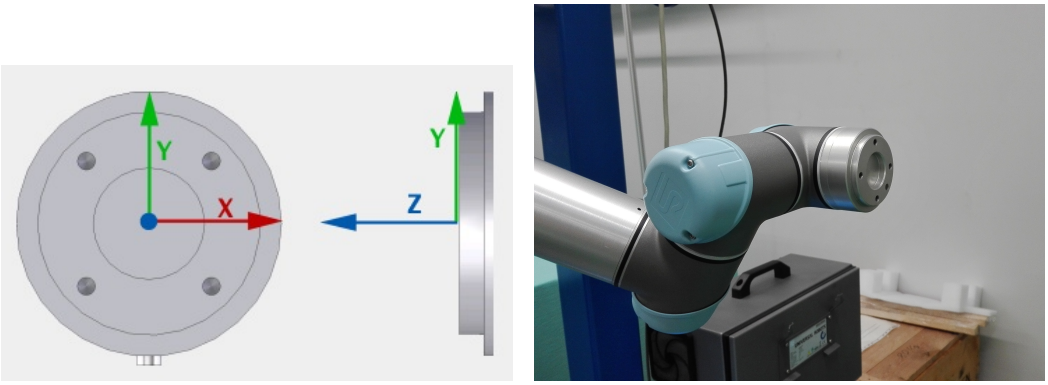


Figura 3.17: Referencial base do elemento terminal do *UR5*

- *set_payload* - Para especificar o peso das ferramentas, sendo 1 kg, para a montagem 1, e 2.3 kg, para a montagem 3.

Na figura 3.18, é apresentado o diagrama de atividade da função *execute_task*, implementada para enviar uma trajetória demonstrada para o *UR5*. Todas as funções para controlar este manipulador, foram desenvolvidas em *C++* e encontram-se disponíveis na secção B.1.1 dos anexos. Para dar início à execução da trajetória, tinha de se carregar o ficheiro de *output* do *6DMimic* com extensão .segs (apresentado posteriormente) e carregar em "Ler pontos e executar movimentos", na interface de teste desenvolvida ilustrada na figura 3.19.

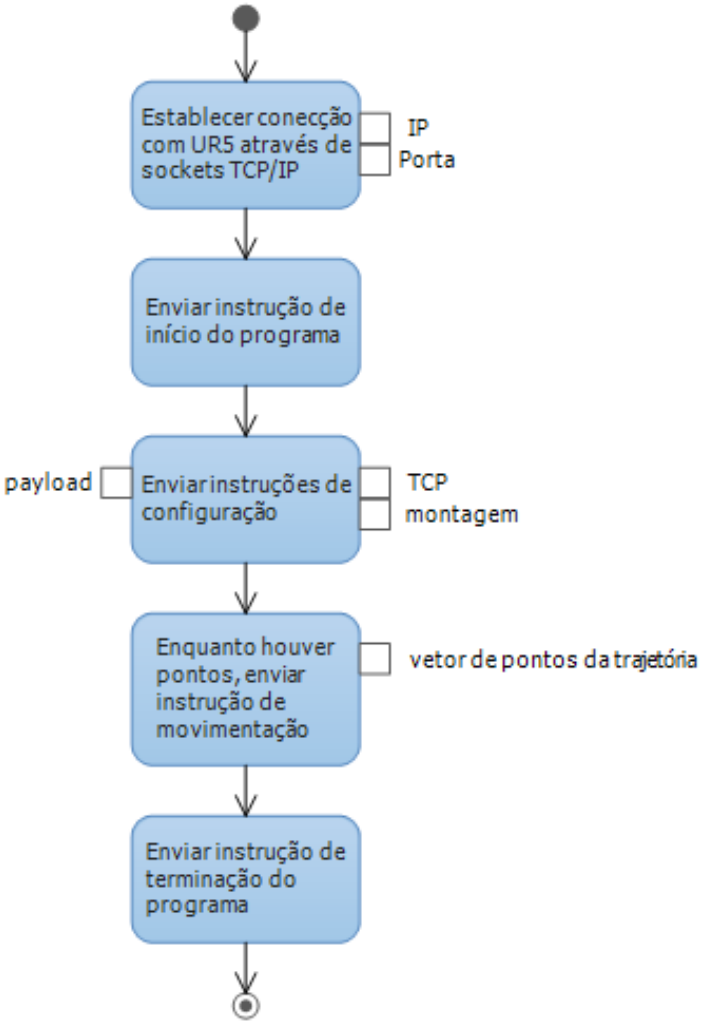


Figura 3.18: Diagrama de atividade do algoritmo implementado para enviar uma trajetória demonstrada para o *UR5*

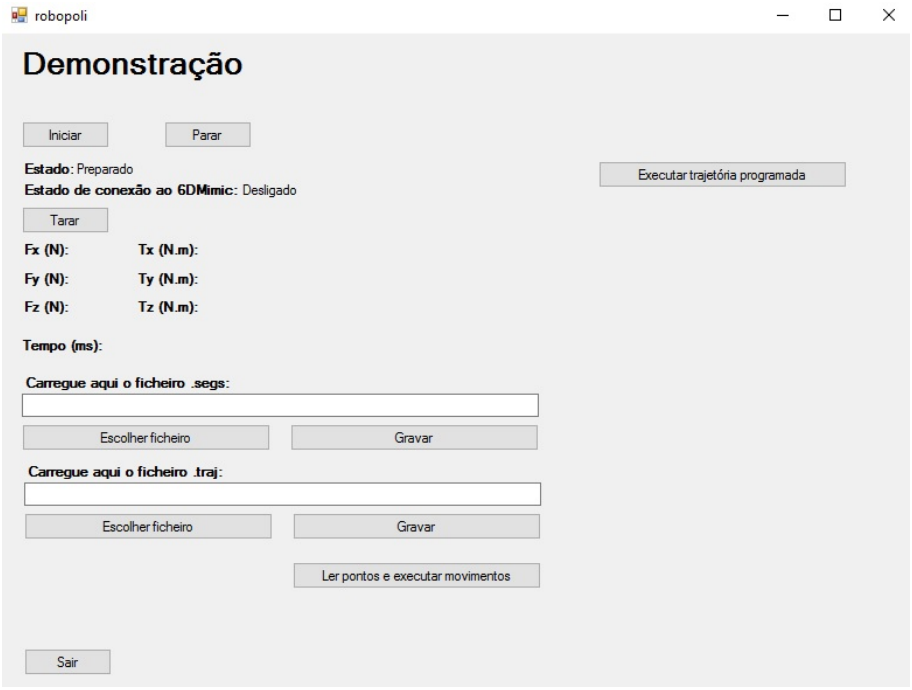


Figura 3.19: Interface de teste desenvolvida para carregar os pontos da trajetória demonstrada e iniciar a sua execução com o *UR5*

3.3.1.2 Leitura do estado

A leitura do estado do *UR5*, foi feita através da descodificação de uma trama, cuja documentação pode ser consultada aqui.

De forma geral, na porta 30002, o controlador deste manipulador vai atualizando uma trama com o formato apresentado na figura 3.20. Esta atualização é feita a uma velocidade de 10 Hz.

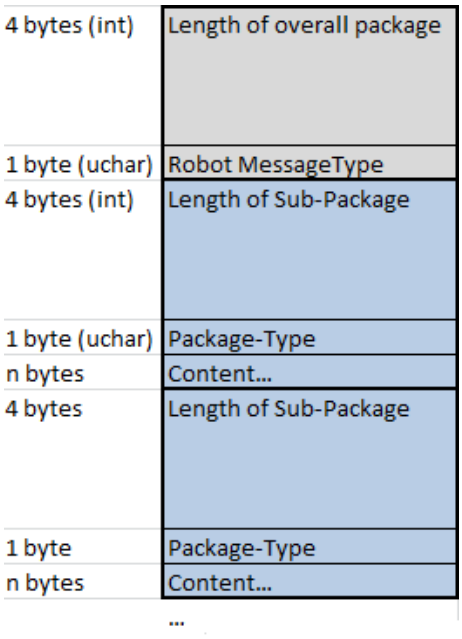


Figura 3.20: Formato da trama com a informação do estado do *UR5*

Quando o "Robot Message Type" tinha o valor 16, significava que a mensagem enviada era do tipo "Robot State" e, dessa forma, poderia ser percorrido o resto da trama à procura de informações sobre o estado do robô.

A tabela seguinte apresenta os possíveis valores e respetivos significados que pode assumir o "Package-Type":

Valor	**Significado**
0	*Robot Mode Data*
1	*Joint Data*
2	*Tool Data*
3	*Masterboard Data*
4	*Cartesian Info*
5	*Kinematics Info*
6	*Configuration Data*
7	*Force Mode Data*
8	*Additional Info*
9	*Calibration Data*

Tabela 3.4: Tipos de pacotes enviados na trama de estado do *UR5*

Onde, por exemplo, num pacote do tipo *Cartesian Info*, temos as seguintes informações sobre a posição e a orientação do referencial da ferramenta, em relação ao referencial base do manipulador:

Tamanho (bytes)	Tipo de dados	Significado
8	double	X (m)
8	double	Y (m)
8	double	Z (m)
8	double	Rx (rad)
8	double	Ry (rad)
8	double	Rz (rad)
...		

Tabela 3.5: Algum conteúdo dentro de um pacote do tipo *Cartesian Info*
Ordem da receção da informação no *buffer*: *TCP/IP network byte order*

Posto isto, desenvolveram-se funções para ler as seguintes informações atuais:

- Pose da ferramenta do manipulador em relação ao referencial base do manipulador - *get_actual_tcp_pose*;
- Posições das articulações - *get_actual_joints_pos*;
- Velocidades das articulações - *get_actual_joints_speed*.

Estas funções, podem ser consultadas na secção B.1.2 dos anexos. O algoritmo desenvolvido resumiu-se à pesquisa das informações pretendidas na trama recebida.

A leitura do estado do manipulador, embora não tenha sido utilizada diretamente nesta parte do projeto, foi bastante útil no desenvolvimento do programa final do sistema automático de inspeção de peças, que será apresentado no capítulo seguinte.

3.3.2 Sensor de força/binário

A comunicação com o sensor de força foi feita através de sockets UDP/IP, segundo o protocolo cuja documentação pode ser consultada no capítulo 10 deste manual. A interface de comunicação disponibilizada usa RDT (*Raw Data Transfer*), o que possibilitou a transferência de dados até 7000 Hz.

A trama de pedido enviada para a *Net Box* tinha o seguinte formato:

Tipo de dados	Signifiado
Uint16	Cabeçalho estático (command_header = 0x1234)
Uint16	Comando a executar (command)
Uint32	Nº de amostras para ler (sample_count)

Tabela 3.6: Formato da trama de pedido enviada para a *Net Box*

Tendo-se enviado os valores command = 0x0002 (*Start high-speed real-time streaming*) e sample_count = 1, quando se pretendeu ler as componentes das forças e binários atuadas no sensor, e o valor command = 0x0042, quando se pretendeu tarar o sensor.

Quando se pretendeu ler as componentes das forças e binários, a trama de resposta tinha o seguinte formato:

Tipo de dados	**Significado**
Uint32	Nº de sequência do pacote (rdt_sequence)
Uint32	Nº da sequência de registo interna (ft_sequence)
Uint32	Código do estado do sistema (status)
Int32	Força no eixo X (Fx) em N
Int32	Força no eixo Y (Fy) em N
Int32	Força no eixo Z (Fz) em N
Int32	Binário no eixo X (Tx) em Nm
Int32	Binário no eixo Y (Ty) em Nm
Int32	Binário no eixo Z (Tz) em Nm

Tabela 3.7: Formato da trama de resposta enviada pela *Net Box*, quando se pretendeu ler as componentes das forças e binários

Quando se pretendeu pretendeu calibrar a tara do sistema, não havia trama de resposta. Com o envio deste pedido, as componentes das forças e dos binários ficavam todas muito próximas de zero.

As forças e binários eram envidas sobre a forma de contagens. Como se configurou 10000000 contagens por N e por Nm (unidades utilizadas para a força e o binário, respetivamente), o valor real das forças foi obtido dividindo os valores recebidos por 10000000. Estabeleceu-se um período de amostragem de 50 ms (20 amostras por segundo).

Todo o código desenvolvido para comunicar com o sensor, é apresentado na secção B.2 dos anexos.

3.3.3 *6DMimic*

A comunicação com o *6DMimic* foi feita através de *sockets* UDP/IP. Desenvolveu-se uma aplicação *Master* (servidor UDP), que enviava os comandos de início e fim da gravação das trajetórias, para a aplicação *Slave* (cliente UDP - *6DMimic*). O IP e as portas da aplicação *Master*, eram configuráveis através da interface gráfica do *6DMimic* (figura 3.21).

Figura 3.21: Interface gráfica do *6DMimic* para configurar a conexão à aplicação *Master*

Nas secções seguintes, será apresentado o protocolo de comunicação estabelecido entre as aplicações *Master* e *Slave* e será explicada a forma como foi feita a sincronização dos relógios destas duas aplicações.

3.3.3.1 Protocolo

As duas tabelas seguintes, apresentam o protocolo de comunicação estabelecido:

Mensagem enviada	Significado
START6DM [file_name]	Pedido enviado para iniciar a gravação das trajetórias. file_name - Nome do ficheiro de *output* .segs.
STOP6DM	Pedido enviado para terminar a gravação das trajetórias.

Tabela 3.8: *6DMimic* - Protocolo de comunicação - Mensagens enviadas pela aplicação *Master*
Nota: Os parêntesis retos não fazem parte das mensagens

Mensagem enviada	Significado
6DM_START [timestamp (microssegundos)]	O *6DMimic* começou a gravar as trajetórias no instante especificado. timestamp (microssegundos) - *Unix Timestamp* em microssegundos
6DM_1sFrame_us [timestamp (microssegundos)]	Envia o instante de tempo em que foi adquirido o primeiro ponto da trajetória. timestamp (microssegundos) - *Unix Timestamp* em microssegundos
6DM_END	O *6DMimic* terminou a gravação das trajetórias sem que tenha ocorrido nenhum erro.
6DM_END_NOK	O *6DMimic* terminou a gravação das trajetórias devido à ocorrência de um erro.

Tabela 3.9: *6DMimic* - Protocolo de comunicação - Mensagens enviadas pela aplicação *Slave*
Nota: Os parêntesis retos não fazem parte das mensagens

O resultado da demonstração de trajetórias, é um ficheiro de *output* com extensão .segs, com uma lista de pontos segundo o seguinte formato:

Tipo de dados	**Posição (mm)**			**Orientação (°)**			**Velocidade do ponto (m/s)**	**Não utilizado**		
Ponto (sempre constante; valor = 'p')	X	Y	Z	Rx	Ry	Rz	v_ponto	-	-	-

Tabela 3.10: Formato de cada linha do ficheiro com os pontos da trajetória
Nota: Os valores são separados por vírgulas.

```
p,43.2556,-591.8347,17.4774,-116.5,0.8,-175.6,1000,0,0,0
p,43.0207,-591.9648,17.4806,-116.4,0.8,-175.5,3,0,0,0
p,42.7997,-592.0122,17.5872,-116.5,0.6,-175.2,3,0,0,0
p,42.1491,-591.6824,17.3861,-116.6,1.1,-175.9,9,0,0,0
p,43.5050,-592.0315,17.5446,-116.4,0.9,-175.1,18,0,0,0
p,42.9260,-591.7545,17.3311,-116.4,1.3,-175.8,8,0,0,0
p,42.1544,-591.7256,17.2000,-116.5,1.6,-175.7,10,0,0,0
p,44.1785,-591.7068,17.3092,-116.5,1.6,-175.7,25,0,0,0
p,43.9685,-591.5837,17.3176,-116.5,1.2,-175.7,3,0,0,0
```

Figura 3.22: Excerto de uma lista de pontos num ficheiro de *output* .segs

A orientação dos referenciais associados aos pontos em relação ao referencial base da demonstração, era dada segundo a notação de rotações de Euler ZYX, em torno do referencial atual.

Como o *UR5* recebia esta orientação segundo a notação Eixo-Ângulo, teve de ser feita a conversão da notação de Euler referida para a notação Eixo-Ângulo. Esta conversão foi feita recorrendo à biblioteca *Eigen*, da seguinte forma:

```
1 //matriz rotacao a partir dos angulos de Euler
2 Rotation_matrix = AngleAxisd( degree_to_rad( Euler_Rz ), Vector3d::UnitZ())
3                 * AngleAxisd( degree_to_rad( Euler_Ry ), Vector3d::UnitY())
4                 * AngleAxisd( degree_to_rad( Euler_Rx ), Vector3d::UnitX());
5
6 //eixo angulo a partir da matriz de rotacao
7 Axis_Angles.fromRotationMatrix(Rotation_matrix);
```

Os pontos listados neste ficheiro, estavam espaçados 80 ms no tempo e as suas coordenadas eram expressas no referencial base de demonstração, como será visto posteriormente.

3.3.3.2 Sincronização de relógios

A memorização dos tempos de aquisição dos pontos das trajetórias e das forças demonstradas era necessária de forma a que, a cada ponto da trajetória, correspondesse uma força. Como a gravação das forças e das trajetórias era feita em dois PCs distintos, era preciso garantir que os relógios de ambos estavam sincronizados.

Posto isto, fez-se a sincronização dos relógios através do algoritmo ilustrado no seguinte diagrama de atividade. Foi ignorada a latência da comunicação, dado o espaçamento temporal entre cada ponto da trajetória (80 ms). As principais funções desenvolvidas para comunicar com o *6DMimic*, estão disponíveis na secção B.3 dos anexos.

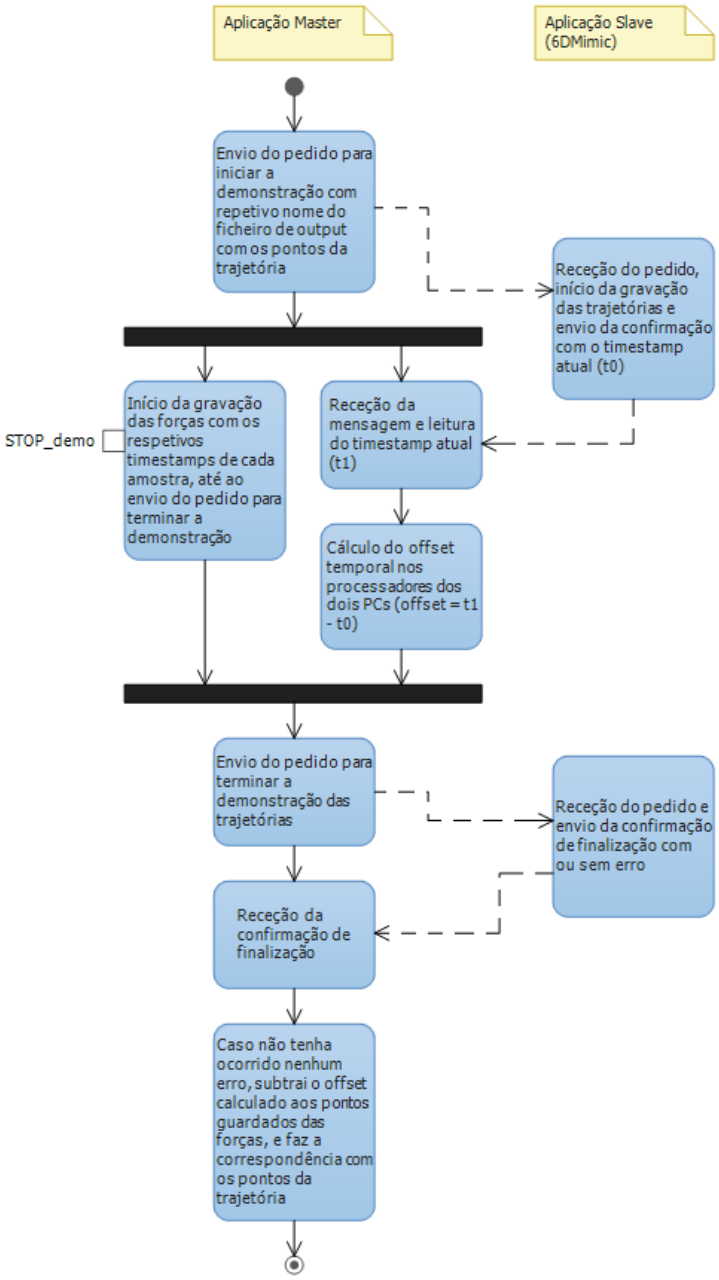


Figura 3.23: Diagrama de atividade do algoritmo implementado para gravar as trajetórias e as forças de forma sincronizada

3.4 Demonstração de trajetórias

A demonstração de trajetórias foi feita com recurso à *framework 6DMimic*, como já foi dito anteriormente.

Nesta secção será apresentada de que forma o sistema foi calibrado, como foram suavizadas as trajetórias e como foram enviadas as instruções para o manipulador industrial *UR5*.

3.4.1 Calibração dos referenciais

Como já foi dito anteriormente, neste sistema de polimento robotizado programado por demonstração humana, existiram duas zonas a considerar: a zona de demonstração da atividade de polimento, onde eram gravadas as forças e as trajetórias envolvidas, e zona de atuação, onde o *UR5* executava o polimento robotizado, a partir das trajetórias e das forças gravadas. Posto isto, uma vez que as duas zonas referidas correspondiam a espaços físicos distintos, o processo de calibração dos referenciais foi extremamente necessário para garantir a execução correta das trajetórias com o *UR5* sobre a polidora.

Esta parte do projeto foi realizada em colaboração com a Ana Sofia Barbosa, no âmbito da sua dissertação cujo tema era "Aplicação robotizada de fibras com programação rápida do manipulador e controlo de qualidade". Desta colaboração, surgiu alguma troca de ideias, que acabaram implementadas da mesma forma em ambos os projetos.

Nas figuras 3.24 e 3.25, são apresentadas as duas zonas referidas, assim como todos os referenciais considerados. O referencial base da demonstração (referencial 0), é coincidente com o referencial base do manipulador industrial da MOTOMAN presente na zona de demonstração. Na tabela 3.11, é explicado o significado de cada referencial considerado.

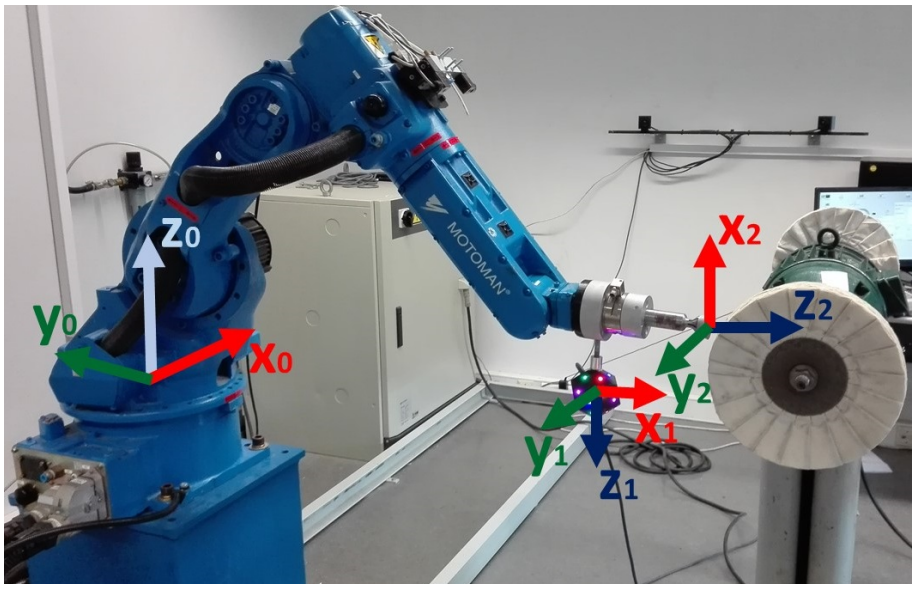


Figura 3.24: Referenciais considerados na zona de demonstração

Figura 3.25: Referenciais considerados na zona de atuação

Nº do referencial	Significado
0	Referencial base da demonstração, coincidente com o referencial base do *MOTOMAN*.
1	Referencial associado à pose do marcador luminoso. Este é o referencial associado aos pontos retornados no ficheiro de *output* com extensão .segs, referido anteriormente. A posição e a orientação deste referencial são expressas segundo o referencial 0.
2	Referencial associado à pose do ponto central da ferramenta. No processo de calibração, a posição e a orientação deste referencial, também são expressas no referencial 0, foram lidas no *teach pendant* do *MOTOMAN*. No processo de demonstração, a posição e a orientação deste referencial são calculadas a partir de uma transformação homogénea aplicada ao referencial 1.
3	Referencial base do *UR5*.

Tabela 3.11: Referenciais considerados nas zonas de demonstração e de atuação

A partir da informação dada na tabela acima, pode resumir-se que o processo de calibração consistiu na descoberta das matrizes de transformação homogéneas:

- Do referencial associado ao ponto central do marcador luminoso (referencial 1), para o referencial associado ao ponto central da ferramenta, dado em relação ao referencial base da demonstração (referencial 2) - H_2^1;

- Do referencial base de demonstração (referencial 0), para o referencial base do *UR5* (referencial 3) - H_0^3.

A figura seguinte, apresenta todas as transformações homogéneas tidas em conta nos cálculos.

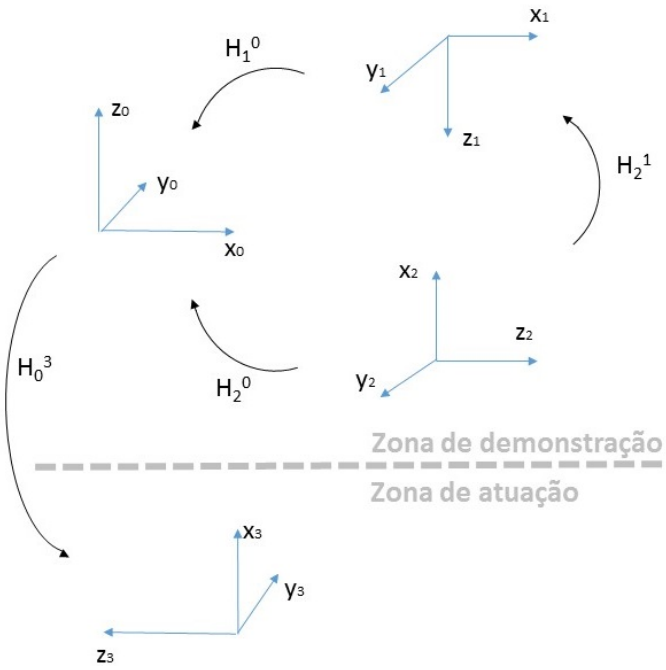


Figura 3.26: Transformações homogéneas envolvidas para transformar os pontos das trajetórias demonstradas para o referencial base do *UR5*

Na figura acima, para além das matrizes H_2^1 e H_0^3, no processo de calibração as matrizes H_1^0 e H_2^0 têm o seguinte significado:

- H_1^0 - Matriz de transformação homogénea formada pela matriz de rotação (R_1^0) e pelo vetor deslocamento (v_1^0), associados ao ponto do marcador luminoso dado pelo ficheiro de *output* do *6DMimic*, no momento de calibração;

- H_2^0 - Matriz de transformação homogénea formada pela matriz de rotação R_2^0 e pelo vetor deslocamento v_2^0, associados ao ponto central da ferramenta dado pelo *teach pendant* do *MOTOMAN* (em relação ao referencial base do manipulador), no momento de calibração. Na aquisição deste ponto, foi necessário ter-se corretamente configurada a ferramenta do manipulador.

Estes pontos de calibração foram adquiridos com a ferramenta orientada perpendicularmente ao disco da polidora, e com o ponto central da ferramenta encostado a um ponto de referência marcado no mesmo disco.

Desta forma, do processo de calibração resultava a matriz H_2^1, dada da seguinte forma:

$$H_2^1 = (H_1^0)^{-1} H_2^0 = H_0^1 H_2^0 \tag{3.1}$$

A matriz H_0^3, foi obtida por inspeção dos referências de base do *MOTOMAN* e do *UR5*. Devido às montagem dos manipuladores, esta matriz é formada pela matriz de rotação (R_0^3) correspondente a uma rotação de 90° em torno do eixo y_3, e pelo vetor de deslocamento nulo.

Por último, uma vez que a distância da polidora à origem dos referenciais base dos manipuladores era diferente, o último passo da calibração consistiu no cálculo de um vetor deslocamento para corrigir este problema. Sendo $v_{UR\_tool}^3$ o vetor deslocamento associado ao ponto central da ferramenta do *UR5* em relação ao seu referencial base, adquirido no momento de calibração, o vetor deslocamento em causa, v, foi calculado da seguinte forma:

$$v = R_0^3 v_2^0 - v_{UR\_tool}^3 \tag{3.2}$$

A partir das matrizes e do vetor calculados, H_2^1, H_0^3 e v, os pontos das trajetórias demonstrados foram corretamente transformados da seguinte forma:

$$H_2^3 = H_0^3 H_2^0 - H_v = H_0^3 H_1^0 H_2^1 - H_v \tag{3.3}$$

Onde H_v é a matriz de transformação homogénea formada pela matriz de rotação nula e pelo vetor deslocamento v, e H_2^3 é a matriz de transformação homogénea formada pela matriz de rotação e o vetor deslocamento, associados ao ponto central da ferramenta em relação ao referencial base do *UR5*. Este último ponto considerado, era o enviado na instrução de movimentação do manipulador.

3.4.2 Suavização

A suavização das trajetórias foi feita através da diminuição do número de instruções enviadas para o *UR5*.

Devido ao facto dos pontos das trajetórias estarem espaçados por apenas 80 ms no tempo, para trajetórias lentas eram geradas demasiadas instruções de movimentação para espaços relativamente curtos. Para piorar a situação, verificou-se que a aquisição dos pontos da trajetória apresentavam um desvio padrão máximo no eixo x do referencial base da demonstração de 0.48 mm. Este facto provocava pequenas vibrações no ponto central da ferramenta, mesmo para uma demonstração parada auxiliada pelo *MOTOMAN*.

Este problema era atenuado com a especificação da tolerância de paragem do ponto central da ferramenta, r, especificado na função *movel*. Contudo, para os movimentos de rotação da ferramenta muito utilizados no polimento, esta tolerância não funcionava pois tratava-se de uma tolerância linear e não angular. Desta forma, para rotações demonstradas de, por exemplo 90°, em

torno do eixo z da ferramenta executadas com alguma velocidade, o *UR5* demorava muito mais tempo a executar essa rotação devido à sobrecarga de instruções enviadas.

O diagrama de atividade seguinte, apresenta o algoritmo implementado para suavizar a trajetória demonstrada, ajustar a sua velocidade e realizar a transformação dos seus pontos expressos no referencial base da demonstração, para o referencial base do UR5. Na secção B.4 dos anexos, está presente o código da função *transform_6DMimic_points*, correspondente ao diagrama de atividade em causa.

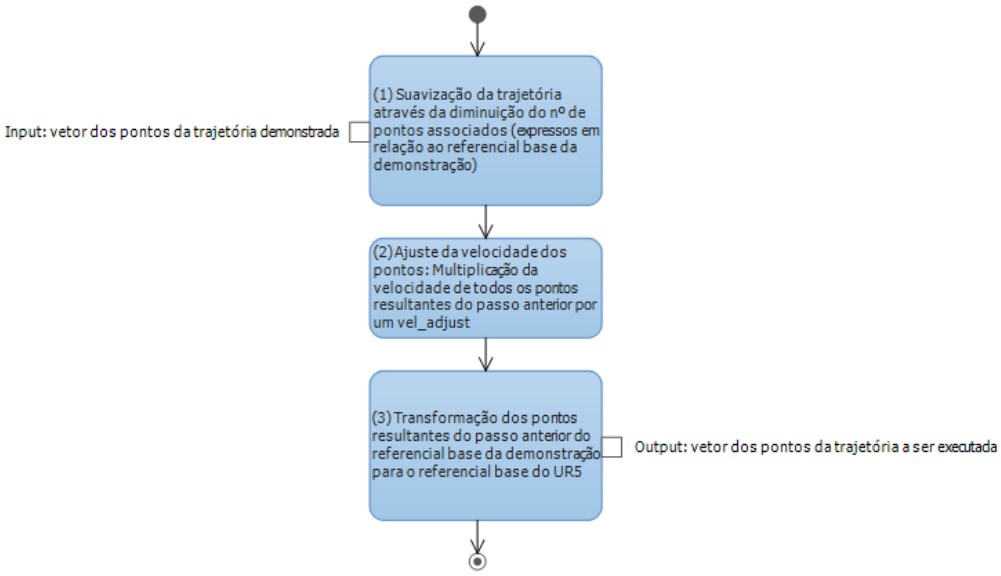


Figura 3.27: Diagrama de atividade do algoritmo implementado para transformar a trajetória demonstrada

A ação 1 do diagrama de atividade acima, correspondente à suavização da trajetória, foi implementada através do seguinte algoritmo:

1. Para começar, é criado um vetor sem nenhum elemento para os pontos da trajetória suavizada;

   Para cada ponto associado à trajetória demonstrada, expresso em relação ao referencial base da demonstração, é feito o seguinte:

2. Se o ponto em causa corresponder ao primeiro ou ao último ponto da trajetória demonstrada, então esse ponto é imediatamente inserido no início ou no fim, respetivamente, do vetor dos pontos da trajetória suavizada. Se isto se verificar, o passo a seguir é ignorado;

3. É calculado o valor absoluto da rotação e do deslocamento entre o referencial associado ao ponto em causa e o referencial associado ao último ponto no vetor dos pontos da trajetória suavizada. Tendo em conta os valores da rotação e do deslocamento obtidos, um dos seguintes passos é executado:

3.1. Se a rotação e o deslocamento forem inferiores a uma tolerância angular e de deslocamento, respetivamente, são acumuladas em variáveis auxiliares as componentes do ponto em causa no que diz respeito à sua velocidade, posição e orientação, assim como é incrementado o número total de pontos acumulados;

3.2. Se a rotação ou o deslocamento forem iguais ou superiores a uma tolerância angular ou de deslocamento, respetivamente, primeiramente são acumuladas as componentes referidas no ponto anterior do ponto em causa, assim como é incrementado o número de pontos acumulados. Posteriormente, são calculadas as médias dessas componentes (referentes a todos os pontos acumulados) e é inserido o ponto médio resultante no vetor dos pontos da trajetória suavizada. Por fim, são colocadas a zero todas as variáveis auxiliares utilizadas.

Para o cálculo do valor absoluto da rotação entre os referenciais, referido no ponto 3, usaram-se quaterniões. Sendo q_1 o quaternião da rotação do referencial associado ao ponto em causa (expresso em relação ao referencial base da demonstração), e q_2 o quaternião da rotação do referencial associado ao último ponto no vetor dos pontos da trajetória suavizada (também expresso em relação ao referencial base da demonstração), o quaternião da rotação entre estes dois referenciais foi calculado da seguinte forma:

$$q_3 = q_2 {q_1}^{-1} \tag{3.4}$$

Para se realizar a comparação dessa rotação com a tolerância angular especificada, utilizou-se a biblioteca *Eigen* para converter o quaternião em notação Eixo-Ângulo (composta por um vetor unitário e um ângulo de rotação em torno desse vetor) e usou-se o ângulo de rotação obtido nesta notação. A acumulação das componentes da orientação dos pontos em causa, também foi realizada com recurso à notação Eixo-Ângulo, onde foram tidas em conta as componentes x, y e z, dos vetores unitários, e os ângulos de rotação em torno desses vetores.

A chamada da função *transform_6DMimic_points* tem o seguinte formato:

```
1 void demo::transform_6DMimic_points(double a, double r, double posToleration,↩
      double angleToleration, double vel_adjust, universal_robot::pose ↩
    p_calib_6DM, universal_robot::pose p_calib_MOTOMAN, universal_robot::pose↩
     p_calib_UR)
```

Onde:

- a é a aceleração dos pontos em m/s^2 - valor usado igual a 4 m/s^2;
- r é a tolerância linear de paragem do ponto central da ferramenta em m - valor usado igual a 0.012 m;

- *posToleration* é o valor absoluto da tolerância de deslocamento mínima (em *m*) referida anteriormente - valor usado igual a 0.015 *m*;
- *angleToleration* é o valor absoluto da tolerância angular (em graus) referida anteriormente - valor usado igual a 7°;
- *vel_adjust* é um parâmetro para aumentar ou diminuir a velocidade de execução da trajetória. Usou-se, por exemplo, o valor 2 para duplicar esta velocidade;
- *p_calib*6*DM*, *p_calib_MOTOMAN* e *p_calib_UR*, são as poses adquiridas no momento de calibração.

Os valores acima usados, foram obtidos através da realização de vários testes, até que a trajetória executada pelo *UR5* correspondesse o mais fidedignamente à trajetória demonstrada pelo operador.

Nota: Os pontos das trajetórias demonstradas (expressos em relação ao referencial base da demonstração) e os das trajetórias a executar (expressos em relação ao referencial de atuação), estavam em vetores de poses associados à classe de demonstração, *demo*, implementada em *C++*.

3.5 Controlo de força

Nesta secção, serão apresentados os modos de controlo de força do *UR5* e os testes realizados ao modo escolhido.

3.5.1 Tipos

Resumidamente, este manipulador pode ser configurado para controlar a força de quatro modos diferentes:

- **Simples:** A força é controlada segundo o eixo z do referencial selecionado;
- **Complexo:** A força é controlada até seis graus de liberdade (Fx, Fy, Fz, Tx, Ty e Tz) segundo o referencial selecionado;
- **Ponto:** O referencial de trabalho tem sempre o eixo y a apontar na direção formada pelo vetor, que aponta desde o ponto central da ferramenta do manipulador, até à origem do referencial selecionado;
- **Movimento:** A força é executada perpendicularmente à direção do movimento.

O modo escolhido foi o complexo, uma vez que se pretendia especificar as componentes da força segundo o referencial da ferramenta.

Para mais informações sobre os modos de controlo de força, deve ser consultado o manual de utilizador indicado na secção 3.2.1.

3.5.2 Testes e resultados

Nesta secção, serão apresentados os testes realizados ao controlo de força do *UR5*, no modo complexo.

Os testes foram realizados colocando as articulações deste manipulador industrial segundo duas configurações distintas, ilustradas nas figuras 3.29 e 3.30. A direção e o sentido das forças exercidas, é a indicada pelo vetor da força, $\vec{F}$.

Utilizou-se o sensor de força/binário apresentado anteriormente, para medir as forças controladas pelo *UR5* no ponto central da ferramenta.

Os referenciais associados ao ponto central da ferramenta do manipulador e do sensor de força, foram configurados para coincidirem com o ponto central da peça polida (referenciais apresentados na figura 3.29 e 3.30).

Utilizou-se uma interface gráfica acessível através do *teach pendant* do *UR5* (figura 3.28), para se realizarem os testes ao controlo de força. A partir desta interface era possível especificar:

- O modo de controlo;
- O referencial tido em conta;
- Os eixos segundo os quais se pretendia aplicar a força;
- Os valores das forças;
- A velocidade de aproximação ao obstáculo.

Esta foi uma maneira prática de testar o controlo de força, sem ser necessário escrever alguma linha de código. O referencial tido em conta, foi o da ferramenta apresentado anteriormente.

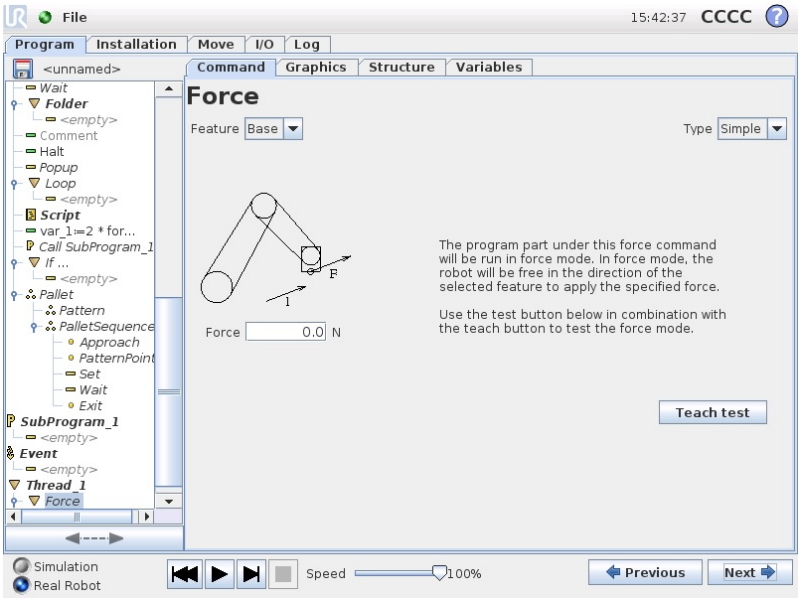
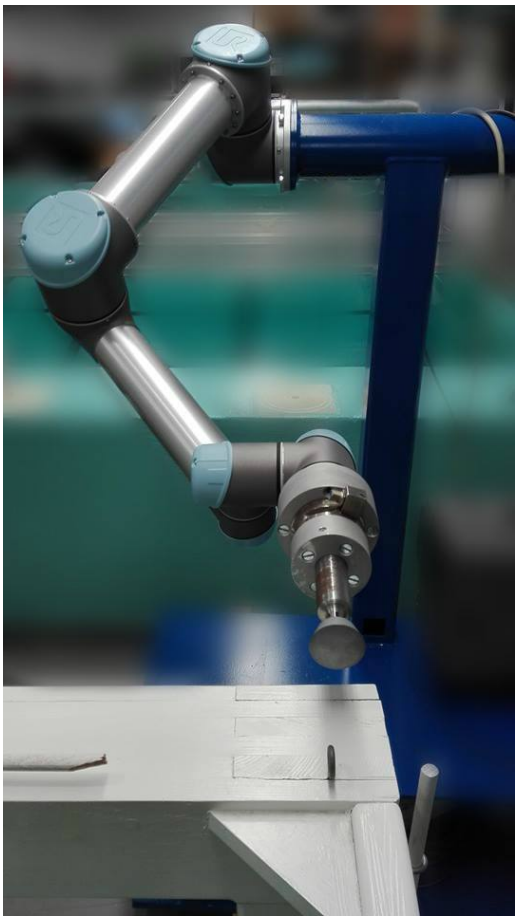
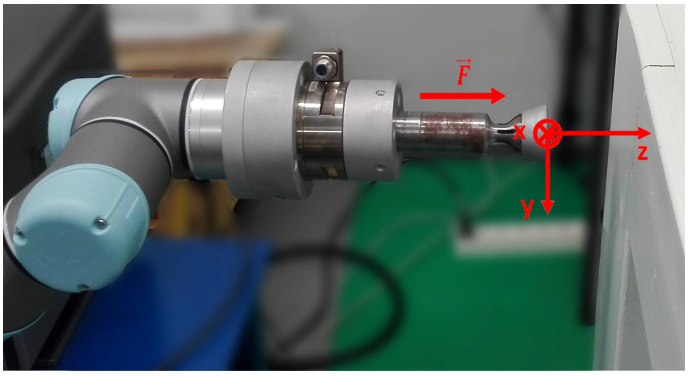


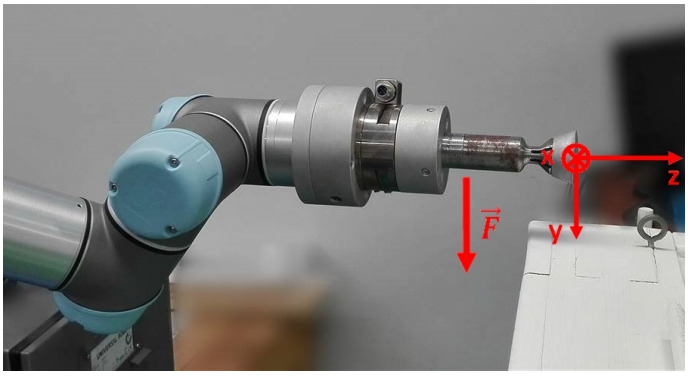
Figura 3.28: Interface gráfica do *teach pendant* do *UR5* para testar o controlo de força



(a) Visão geral



(b) Teste do eixo z do referencial da ferramenta



(c) Teste do eixo y do referencial da ferramenta

Figura 3.29: Configuração 1 - Testes efetuados contra o poste e a trave de uma baliza

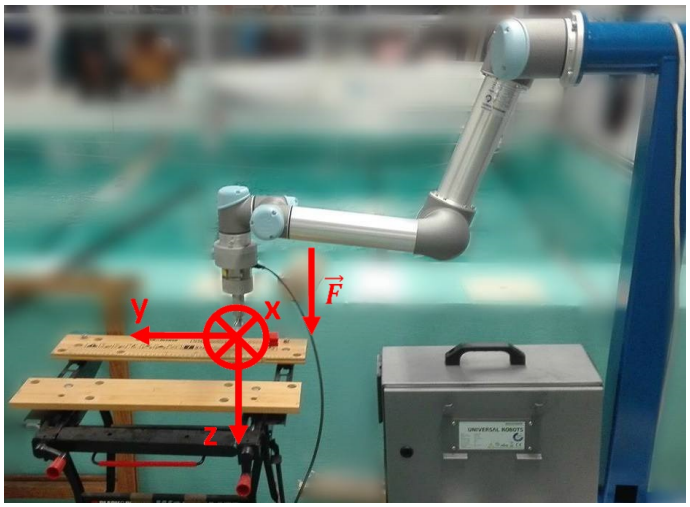


Figura 3.30: Configuração 2 - Testes efetuados contra uma mesa - Teste do eixo z do referencial da ferramenta

Posto isto, com todas as componentes do binário a zero, assim como as da força, com exceção do eixo que se queria testar, e com uma velocidade de aproximação muito baixa para evitar forças muito elevadas no embate contra os obstáculos, obtiveram-se os resultados apresentados de seguida.

Nota: Antes de se iniciar o movimento de aproximação aos obstáculos, orientou-se o eixo a testar perpendicularmente aos últimos e calibrou-se a tara do sensor de força para ignorar o peso da ferramenta.

Configuração 1:

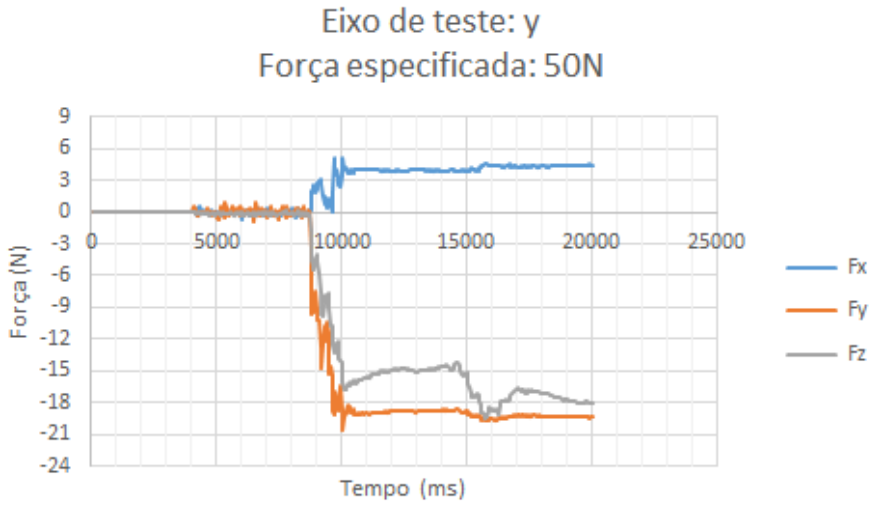


Figura 3.31: Configuração de teste: 1
Eixo: y
Força: 50N

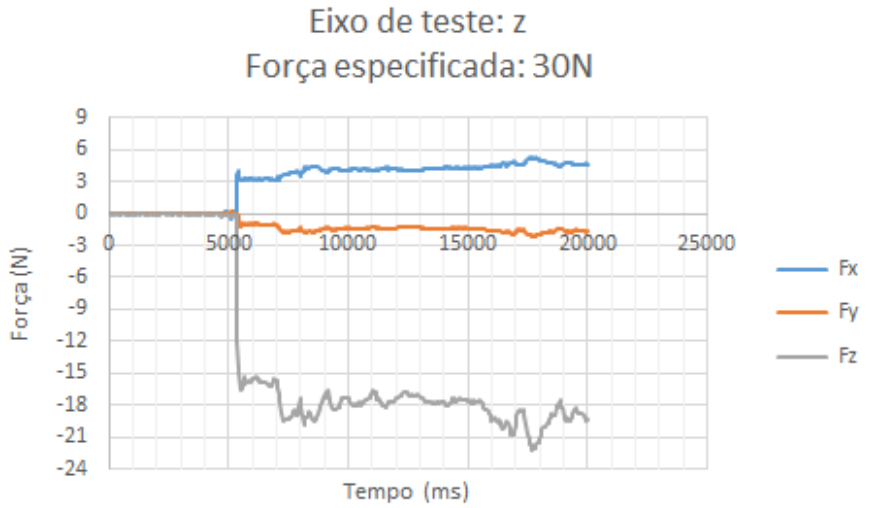


Figura 3.32: Configuração de teste: 1
Eixo: z
Força: 30N

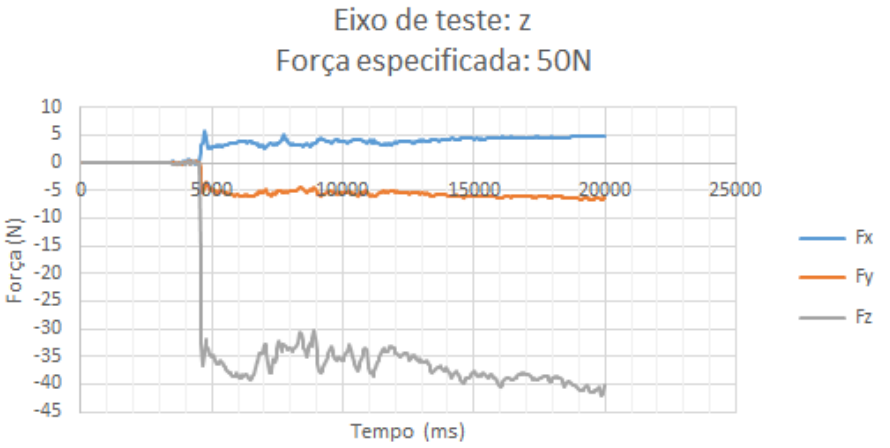


Figura 3.33: Configuração de teste: 1
Eixo: z
Força: 50N

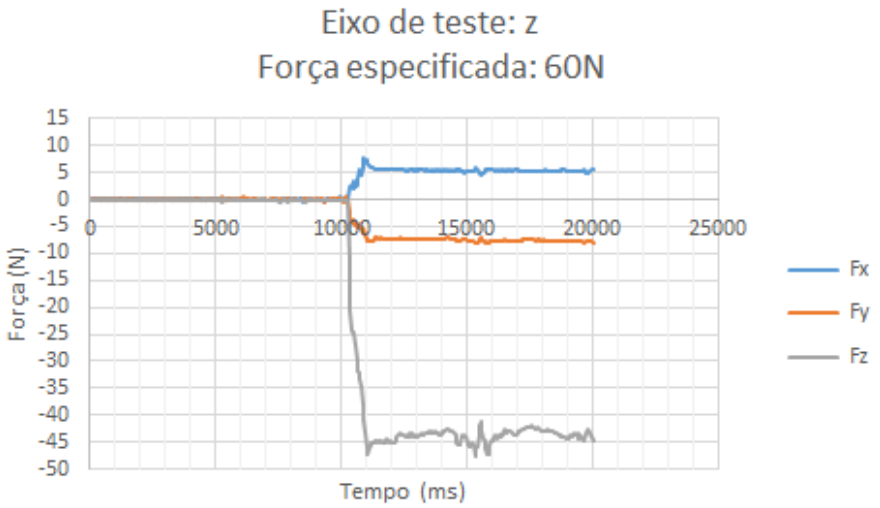


Figura 3.34: Configuração de teste: 1
Eixo: z
Força: 60N

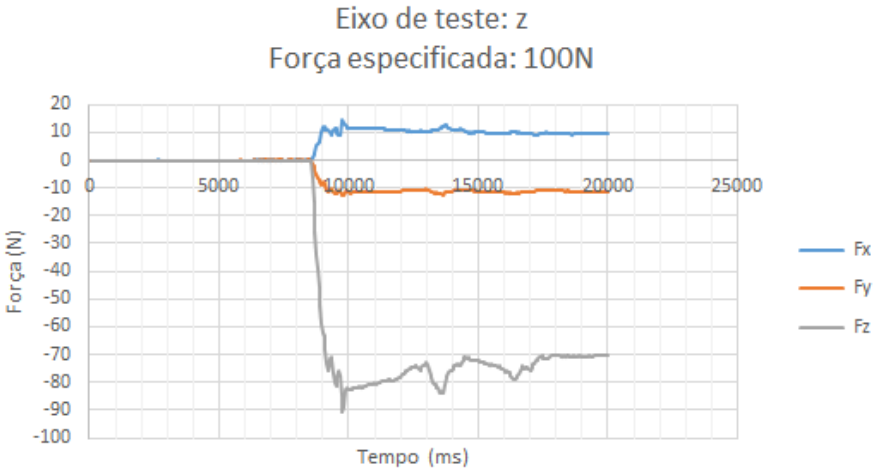


Figura 3.35: Configuração de teste: 1
Eixo: z
Força: 100N

Configuração 2:

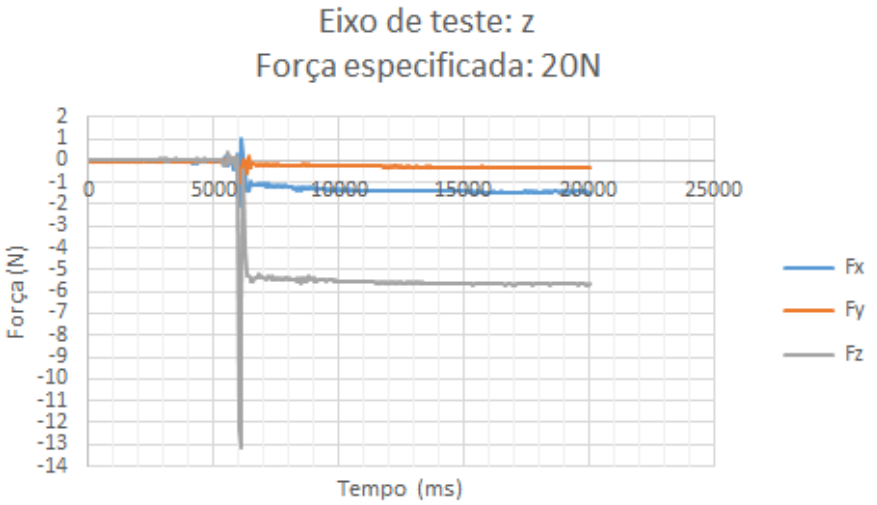


Figura 3.36: Configuração de teste: 2
Eixo: z
Força: 20N

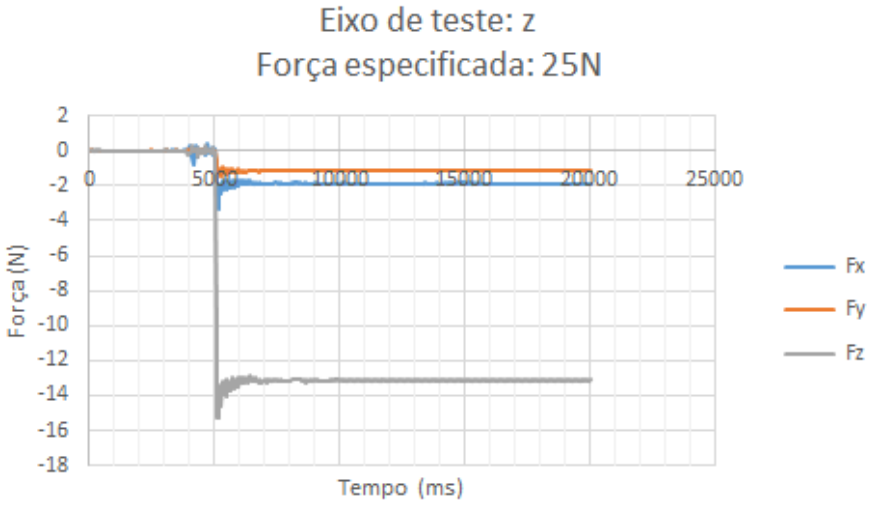


Figura 3.37: Configuração de teste: 2
Eixo: z
Força: 25N

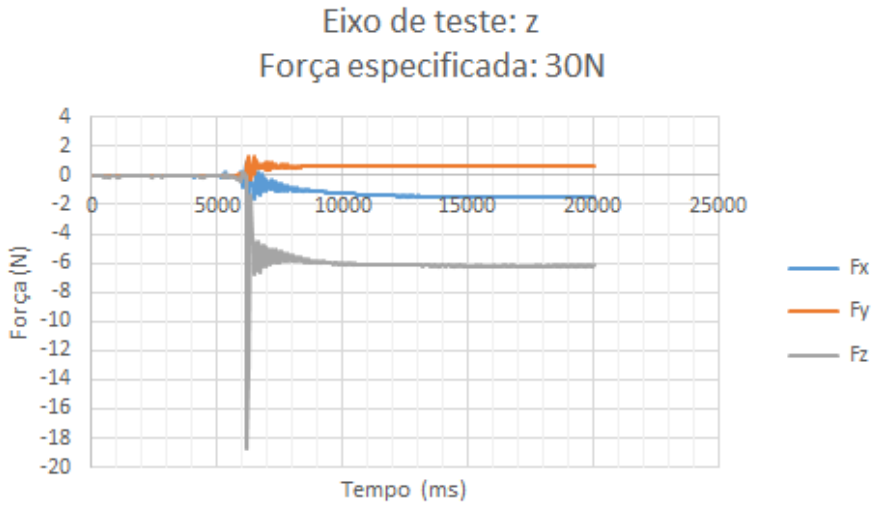


Figura 3.38: Configuração de teste: 2
Eixo: z
Força: 30N

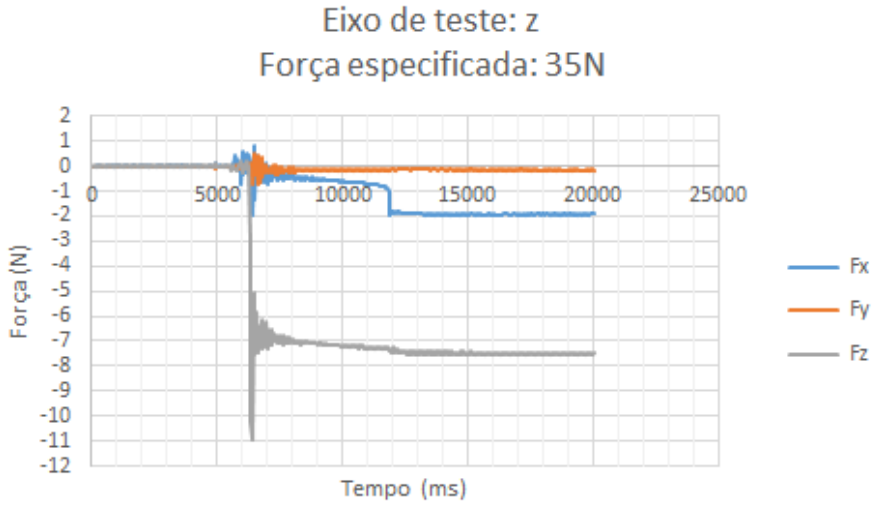


Figura 3.39: Configuração de teste: 2
Eixo: z
Força: 35N

3.5.3 Conclusões

A partir dos resultados apresentados, pôde concluir-se que as posições das articulações do manipulador influenciavam claramente a qualidade do controlo de força executado. Com a configuração 1 foram obtidos os melhores resultados, contudo, verificaram-se os seguintes problemas:

- Após o impacto com o obstáculo, o valor final da força parecia não conseguir estabilizar e oscilava dentro de uma gama de valores relativamente alta. Por exemplo, para o teste do eixo z com 100 N, o valor final oscilava entre -90 e -70 N;
- Em relação ao valor da força especificada, o valor final obtido tem um *offset* que tendia a aumentar à medida que se aumentava a força especificada;
- Para o teste feito ao eixo y com 50 N, foi bem percetível o espelhamento da força sobre o eixo z. Contudo, este facto estava relacionado com a mecânica do sensor utilizado que, para o movimento efetuado, não se conseguia evitar.

Para a configuração 2 verificou-se que:

- O manipulador não conseguia controlar tão bem a força de impacto;
- Embora a força final obtida fosse mais estável (possivelmente devido às elasticidades dos obstáculo, embora não existisse grande evidência desse facto), verificou-se que à medida que se aumentava a força especificada, mediam-se forças finais aleatórias entre -5 e -14 N. Este facto deu a entender que, para esta configuração, a força não era de todo controlada.

A partir dos factos apresentados, concluiu-se que, para aplicações que não requerem controlo de força de alta precisão, o controlo de força do *UR5* sem o uso de um sensor de força externo pode ser considerado. Caso contrário, deverá ser usado um sensor de força externo. No caso concreto do polimento, deverá ser testado se um controlo de força pouco preciso oferece bons resultados finais.

O problema identificado do *offset* da força em relação à especificada, pode ser corrigido através de uma curva de calibração obtida para uma certa posição das articulações.

Capítulo 4

Inspeção por visão artificial de peças altamente refletoras e espelhadas

4.1 Visão geral da solução proposta

Nesta secção pretende-se descrever uma visão geral da solução proposta face ao problema concreto de inspeção por visão artificial desta dissertação.

Tinha-se como objetivo principal desenvolver um sistema o mais genérico possível, que permitisse a inspeção de qualquer tipo de peça com superfície altamente refletora e espelhada, obviamente tendo em conta restrições como a forma e o tamanho das peças, assim como o tipo de defeitos típicos que pretendem ser detetados.

Para efeitos de teste, considerou-se um conjunto de peças pintadas a dourado fornecidas pela empresa Lobo & Filhos LTD. A figura 4.1 apresenta as peças referidas, tendo sido feita uma divisão em dois grupos:

- Peças boas (à esquerda das peças polidas no centro do tabuleiro da figura) – Peças que do ponto de vista da empresa referida, não apresentavam defeitos suficientes para que fossem rejeitadas;

- Peças más (à direita das peças polidas no centro do tabuleiro da figura) - Peças que do ponto de vista da empresa referida, apresentavam defeitos suficientes para que fossem rejeitadas.

Figura 4.1: Conjunto de peças pintadas a dourado e polidas com superfície altamente refletora e espelhada

Na secção 4.3 serão apresentados os defeitos tipicamente encontrados nas peças consideradas.

Posto isto, desenvolveu-se um sistema baseado em visão constituído pelos seguintes subsistemas:

- Condicionamento do ambiente de aquisição;
- Aquisição da imagem;
- Manipulação das peças;
- Algoritmo de deteção de defeitos.

O condicionamento do ambiente de aquisição foi feito com recurso a um sistema de visão artificial, que será apresentado na secção seguinte. A iluminação difusa fornecida e o isolamento com o ambiente circundante, forneceram condições para que os defeitos na superfície das peças fossem suficientemente realçados, antes das imagens serem processadas.

A aquisição das imagens foi feita com recurso a uma câmara monocromática, com uma lente, que também fazem parte do sistema de visão artificial, referido no parágrafo anterior.

Usou-se o manipulador industrial apresentado no capítulo anterior, para levar as peças até à zona de inspeção e ser feito o varrimento da superfície completa das peças.

O algoritmo de deteção de defeitos é composto pelo pré-processamento da imagem, segmentação e extração dos defeitos. Escreveu-se este algoritmo na linguagem de programação *C++* utilizando a versão 3.0.0 da biblioteca *OpenCV* (*Open Source Computer Vision Library*), sendo o programa executado num computador pessoal com as caraterísticas apresentadas na tabela 3.3.

Na figura seguinte é apresentada uma visão geral do sistema em causa, que nas próximas secções será descrito detalhadamente.

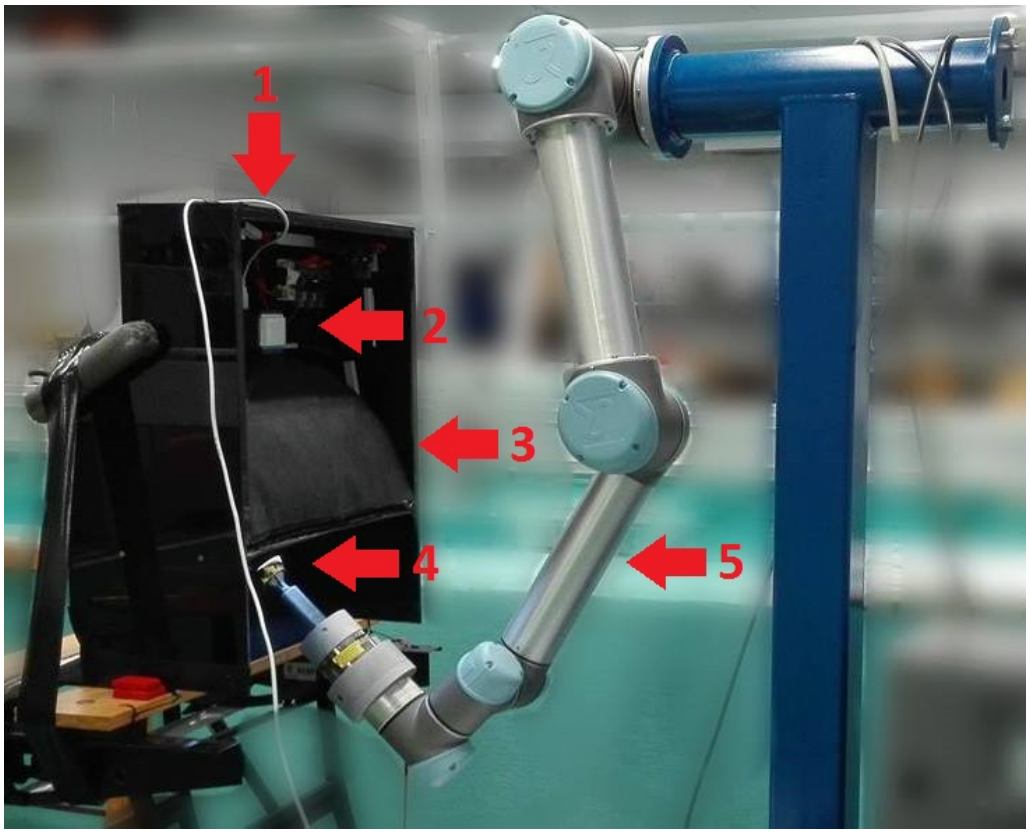


Figura 4.2: Visão geral do sistema de inspeção automática por visão artificial (1 - Sistema de visão artificial; 2 - Câmara e lente; 3 - Cúpula de iluminação difusa; 4 - Peça a ser inspecionada na zona de inspeção; 5 - Manipulador industrial - *UR5*)

4.2 Sistema de visão artificial

Adaptou-se um sistema de visão artificial com uma cúpula de iluminação difusa desenvolvido pelo INESC TEC e melhorado por Roberto Silva, no âmbito da sua dissertação de mestrado [8], para condicionar o ambiente de aquisição. Este sistema é composto pelos seguintes elementos principais:

- Uma caixa com estrutura em acrílico preto, forrada no interior com veludo preto autocolante, para eliminar as reflexões de luz (figuras 4.3 e 4.4);
- Cúpula de iluminação difusa composta por LEDs de alto brilho de cor branca (figura 4.5);
- Placa reguladora da intensidade de iluminação (figura 4.6);
- Sistema de ventilação com placa de controlo manual (para ligar e desligar partes desse sistema) (figura 4.7);
- Câmara (figura 4.8);
- Lente.

Figura 4.3: Sistema de visão artificial - Exterior da caixa [8]

Figura 4.4: Sistema de visão artificial - Interior da caixa [8]

Figura 4.5: Sistema de visão artificial - Cúpula de iluminação difusa [8]

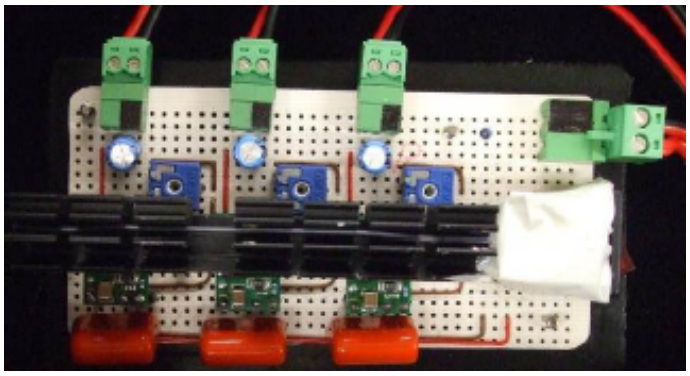


Figura 4.6: Sistema de visão artificial - Placa reguladora da intensidade de iluminação [8]

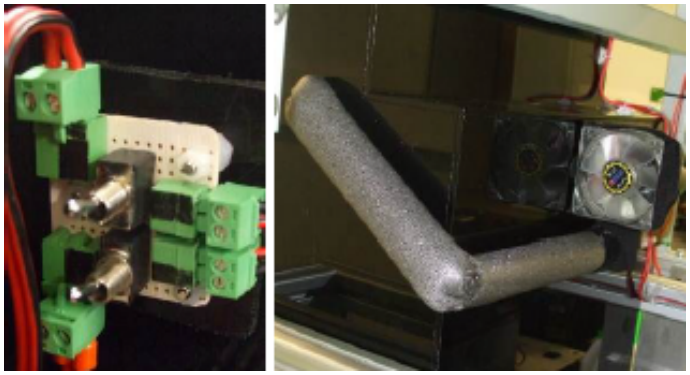


Figura 4.7: Sistema de visão artificial - Sistema de ventilação e respetiva placa de controlo manual [8]

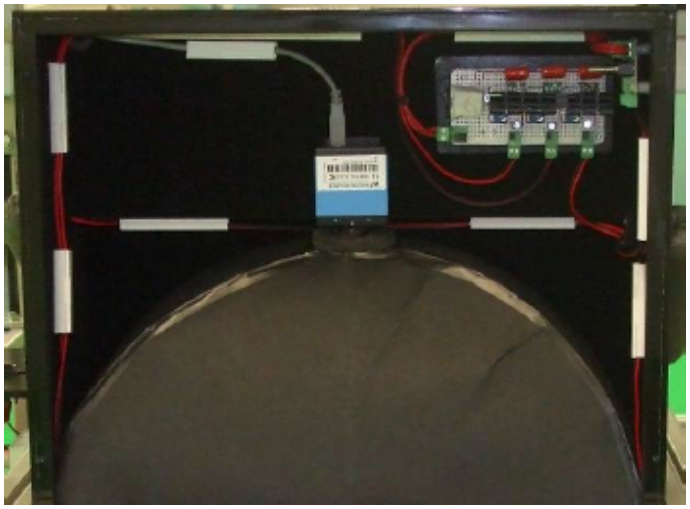


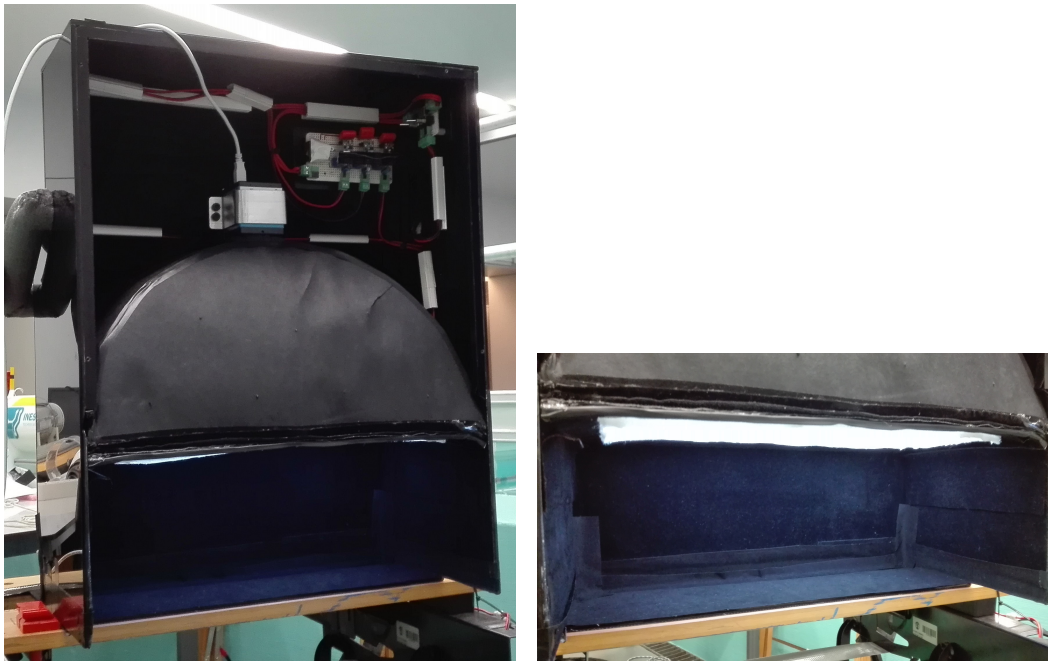
Figura 4.8: Sistema de visão artificial - Local de encaixe da câmara e respetiva lente [8]

O sistema de visão artificial é alimentado a 12V e na sua intensidade máxima consome aproximadamente 1.17A (14,04W). Para alimentar o seu circuito, usou-se uma fonte de alimentação de bancada que preenchesse os requisitos anteriores.

Nas secções seguintes serão justificadas as alterações efetuadas a este sistema.

4.2.1 Zona de inspeção

A zona de inspeção da caixa teve necessariamente de ser adaptada para que fosse possível colocar as peças numa posição que permitisse a sua inspeção através do manipulador *UR5*. Desta forma, abriu-se a parte frontal da caixa, junto à zona de inspeção, e preencheu-se metade do seu fundo com uma placa de acrílico preto revestida com veludo preto. A última alteração está relacionada com a técnica de iluminação que foi usada e que será explicada posteriormente. As figuras 4.9a e 4.9b ilustram as alterações referidas.



(a) (b)

Figura 4.9: Sistema de visão artificial – Adaptação da parte frontal da caixa

4.2.2 Câmara

Utilizou-se a câmara monocromática DMK 31AU03.AS da *The Imaging Source* (figura 4.10), cujas especificações estão presentes na tabela 4.1. Esta era a câmara que vinha com o sistema de visão artificial e, uma vez que se obteve bons resultados com a mesma, não houve necessidade de alteração.

Figura 4.10: *The Imaging Source DMK 31AU03.AS*

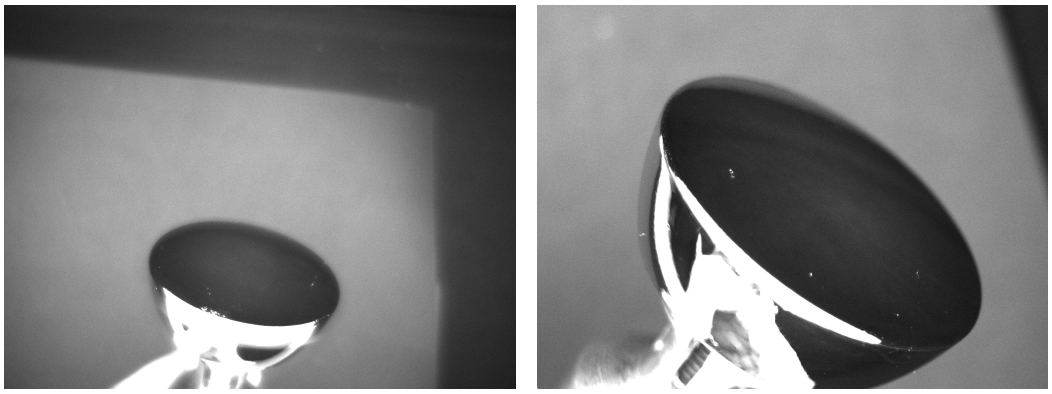
Formatos de vídeo @ FPS	1024 x 768, Y800 @ 30, 15, 7.5, 3.75 FPS
Gama dinâmica	8 bits
Filtro de corte infravermelho	Não
CCD	Sony ICX204AL
Tipo	Varrimento progressivo
Formato	1/3"
Resolução	H: 1024 pixel, V: 768 pixel
Montagem da lente	C/CS
Tipo de conetor	USB
Tensão de alimentação	4.5 VDC até 5.5 VDC (via cabo USB)
Dimensões	H: 50.6 mm, W: 50.6 mm, L: 50 mm
Massa	265 g
Tempo de exposição	1/10,000 segundos até 60 minutos
Ganho	0 dB até 36 dB
Temperatura máxima (operação)	-5 °C até 45 °C
Temperatura máxima (repouso)	-20 °C até 60 °C
Humidade máxima (operação)	20 % até 80 % sem condensação
Humidade máxima (repouso)	20 % até 95 % sem condensação
Folha de especificações	Clique aqui

Tabela 4.1: *The Imaging Source DMK 31AU03.AS* - Especificações

4.2.3 Lente

No que diz respeito à lente, houve necessidade de se fazer uma adaptação e usar uma lente com maior distância de focagem. A lente que estava a ser usada era a LTC 3361/30 da PHILIPS (clique aqui para ver a folha de especificações), cuja distância de focagem era no máximo 8 mm, tendo sido feita a alteração para a lente GMHR31614MCN, da GOYO OPTICAL INC., com 16mm. Na tabela 4.2, encontram-se as especificações da última lente. As figuras 4.11a e 4.11b correspondem a imagens retiradas em condições idênticas com a lente com distância de focagem de 8mm e com

a de 16mm, respetivamente. Com esta alteração foi possível obter-se uma maior resolução da superfície da peça, realçando os defeitos nela contidos.



(a) PHILIPS LTC 3361/30 (b) GOYO OPTICAL GMHR31614MCN

Figura 4.11: Comparação de imagens captadas com as duas lentes

Abertura (F)	1.4
Tipo	Manual
Rosca	C
Distância de focagem	16 mm
Tamanho do sensor	2/3"
Folha de especificações	Clique aqui

Tabela 4.2: GOYO OPTICAL INC. GMHR31614MCN - Especificações

4.3 Defeitos típicos

As figuras seguintes apresentam os vários tipos de defeitos observados nas peças pintadas de teste. Estes deveram-se principalmente a erros no processo de pintura e secagem das peças, ou então a problemas ocorridos durante o transporte das mesmas. Como se pode reparar nas figuras, os defeitos estavam relacionados a saliências e concavidade, e/ou a perda significativa de refletividade e de espelhamento.

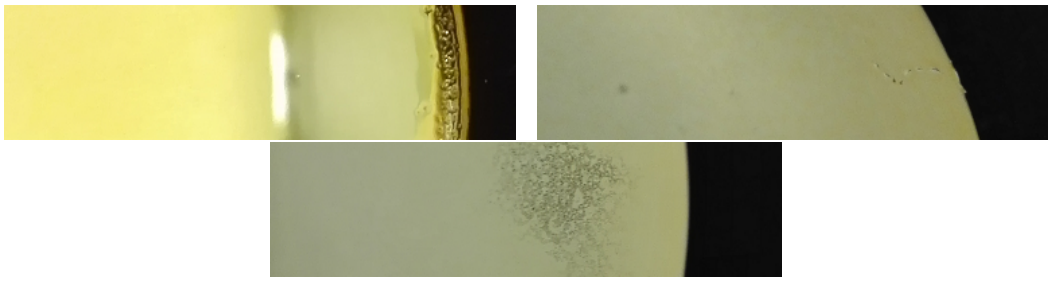


Figura 4.12: Defeitos tipicamente observados na superfície superior da peça

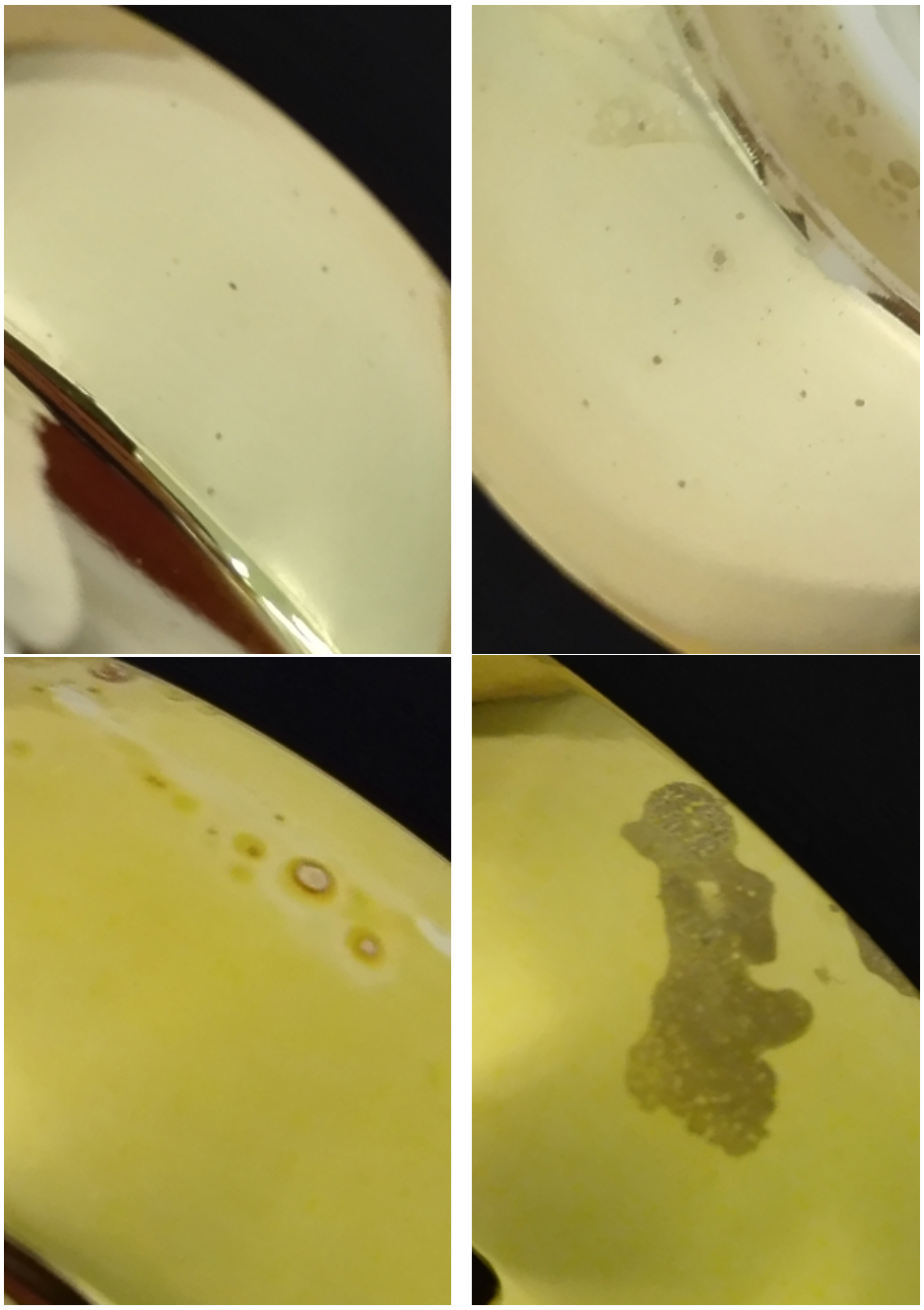


Figura 4.13: Defeitos tipicamente observados na superfície lateral da peça

4.4 Técnica de iluminação

A iluminação difusa é por definição a indicada para inspecionar peças altamente refletoras e espelhadas, como já foi explicado na secção 2.3.3. Esta foi a técnica de iluminação utilizada no que diz respeito à direção da luz, tendo sido usada para o efeito a cúpula de iluminação difusa referida anteriormente.

No sentido de se obter uma imagem idêntica às apresentadas na figura 2.15, foram feitas algumas experiências variando a intensidade da iluminação (através da placa reguladora da intensidade de iluminação apresentada anteriormente), o tempo de exposição da câmara e a posição das peças na zona de inspeção. Note-se que, ao aumentar o tempo de exposição da câmara, maior é a quantidade de luz captada do ambiente, ficando desta forma a imagem mais clara.

A figura 4.14 corresponde a uma imagem captada com a intensidade da iluminação no mínimo, o tempo de exposição da câmara de 1/25 s e a superfície superior da peça perpendicularmente à câmara e à fonte de iluminação. Pode observar-se que, devido ao facto da câmara ser concêntrica com a fonte de iluminação, a quantidade de luz incidente na peça e refletida para a câmara é tanta que os pixeis na zona da peça saturam no máximo (reflexão especular), impossibilitando a extração de qualquer caraterística.

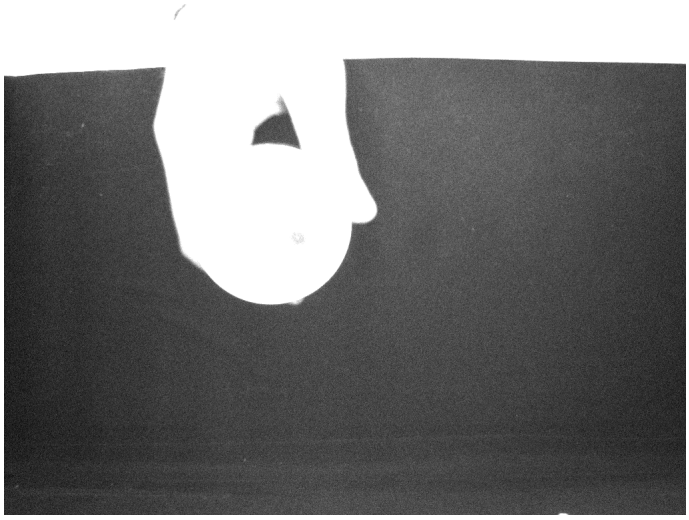


Figura 4.14: Imagem retirada com a intensidade da iluminação no mínimo, tempo de exposição de 1/25 s e a superfície superior da peça perpendicularmente à câmara e à fonte de iluminação

Posto isto e uma vez que a intensidade da iluminação já estava no mínimo, diminui-se o tempo de exposição da câmara até os pixeis deixarem de estar saturados, tendo-se obtido a imagem da figura 4.15. Esta alternativa não se tornou viável devido ao seguintes motivos:

- É espelhado o interior da cúpula difusa (apresentado na figura 4.5) que, ao ser irregular, provoca variações elevadas nas intensidades dos pixeis na superfície da peça;
- A lente da câmara é espelhada, apresentando-se como uma mancha negra na superfície da peça;
- O ruído gaussiano é acentuado, confundindo-se facilmente com os defeitos mais pequenos.

Desta forma, conclui-se que a posição da peça apresentada não era viável.

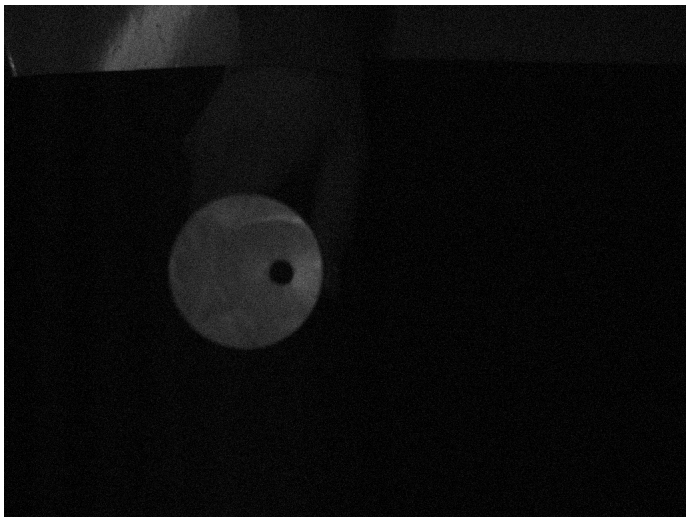


Figura 4.15: Imagem retirada com a intensidade da iluminação no mínimo, tempo de exposição de 1/250 s e a superfície superior da peça perpendicularmente à câmara e à fonte de iluminação

Posicionando a superfície superior da peça obliquamente à câmara e à iluminação, obteve-se o resultado da figura 4.16 (com a intensidade da iluminação no mínimo e um tempo de exposição da câmara de 1/120 s).

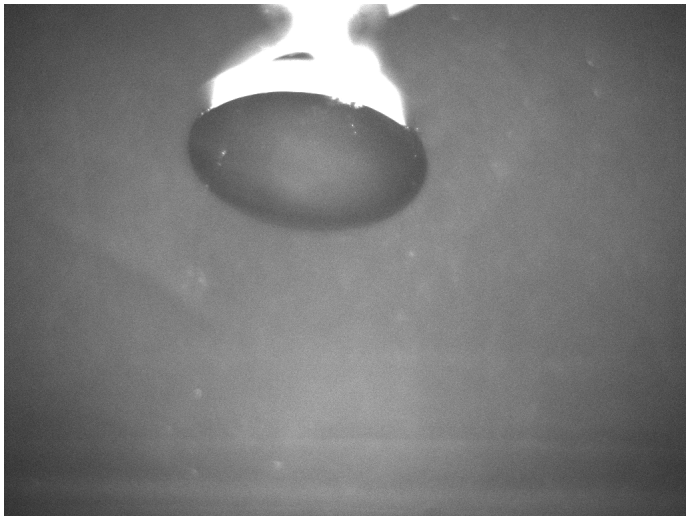


Figura 4.16: Imagem retirada com a intensidade da iluminação no mínimo, tempo de exposição de 1/120 s e a superfície superior da peça obliquamente à câmara e à fonte de iluminação

Esta opção mostrou-se viável devido aos seguintes factos:

- Alguns defeitos típicos são saliências ou concavidades, tendo tornado a utilização de iluminação difusa de campo escuro viável. Recorde-se que, por definição, a iluminação de campo escuro é usada para detetar saliências e concavidades, como foi apresentado na secção 2.3.3;

- A superfície uniforme do veludo de revestimento do interior da caixa do sistema de visão artificial (apresentado na figura 4.4 e reforçado na adaptação feita ilustrada na figura 4.9), mostrou-se viável para espelhar na superfície das peças, uma vez que não provoca variações elevadas e bruscas nas intensidades dos pixeis;
- Alguns defeitos típicos estão relacionados com a perda da refletividade e do espelhamento nessas zonas, não sendo corretamente espelhada a superfície uniforme do veludo referido no tópico anterior, facilitando a deteção dos defeitos.

Esta técnica, contudo, apresenta reflexões especulares em zonas problemáticas da peça, como a borda da superfície superior e uma parte da superfície lateral, devido à posição dessas zonas em relação à iluminação/câmara. Tentou eliminar-se este efeito, novamente, diminuindo o tempo de exposição da câmara, no entanto, observou-se que à medida que se diminuía esse tempo, os defeitos perdiam realce e passavam a ser indetetáveis (figura 4.17). Contrariamente, com o tempo de exposição em 1/120 s e aumentando a intensidade da luz, observou-se que a zona afetada pela reflexão especular não aumentou significativamente (em relação à zona obtida numa imagem com os defeitos suficientemente realçados) e que os defeitos foram realçados ainda mais(figuras 4.18a e 4.18b).

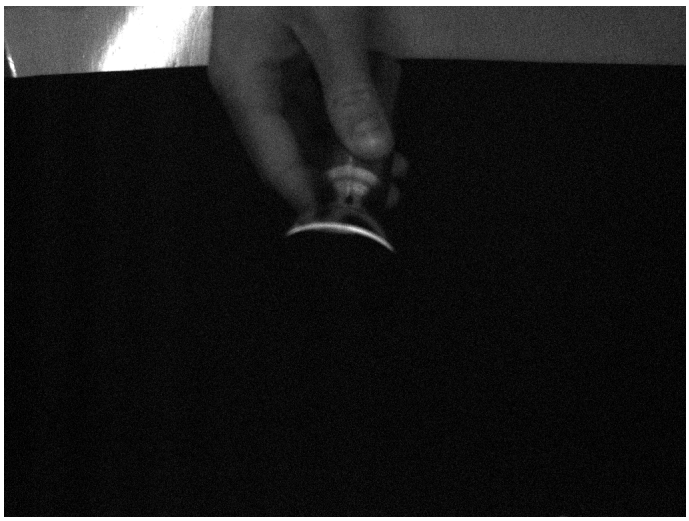
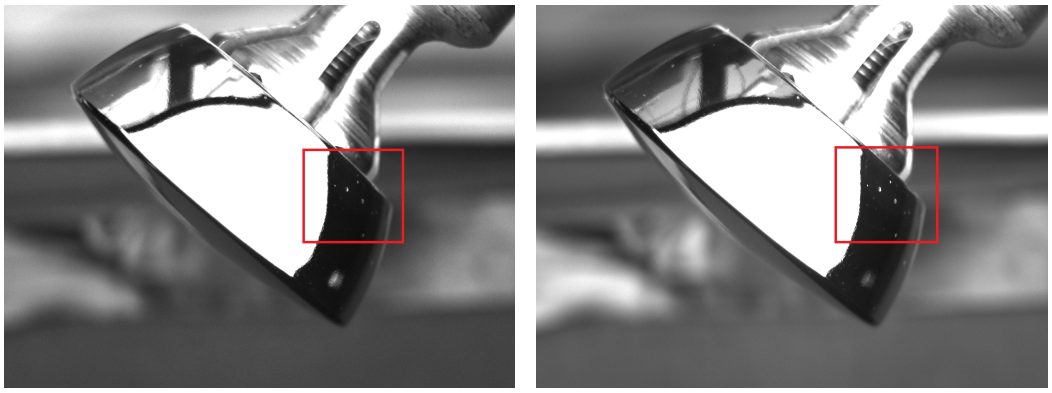


Figura 4.17: Imagem retirada com a intensidade da iluminação no mínimo, tempo de exposição de 1/250 s e a superfície superior da peça obliquamente à câmara e à cúpula de iluminação difusa

Para se obter a imagem final, ajustou-se a posição e a inclinação da peça, o foco da lente e encontrou-se um ponto de equilíbrio entre a intensidade da iluminação e o tempo de exposição da câmara. Neste último ponto, teve-se em conta o facto de que as imagens seriam adquiridas com a peça em movimento e portanto, o tempo de exposição da câmara teria de ser baixo para evitar o rasto deixado pela peça na imagem. Encontrou-se o melhor resultado para um tempo de exposição de 1/120 s e uma intensidade de iluminação correspondente ao sistema de visão artificial a consumir 1.17A (14,04W/12V). No entanto, optou-se por usar uma intensidade mais baixa, o

correspondente a 0.40A (4.8W/12V), uma vez que uma intensidade superior a esta apenas realça defeitos muito pequenos que mais tarde seriam rejeitados pelo algoritmo de deteção. Desta forma, não se justificava estar a consumir uma potência superior a 4.8W. A figura 4.18 faz o comparativo entre estas duas situações, estando assinalada uma zona onde o caso referido acontece.

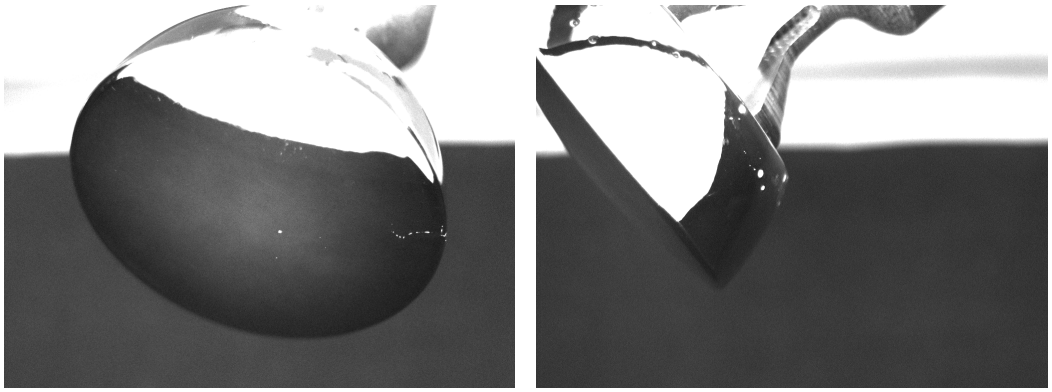


(a) Imagem retirada com uma intensidade de iluminação correspondente ao sistema de visão artificial a consumir 0.40A (4,8W/12V)

(b) Imagem retirada com uma intensidade de iluminação correspondente ao sistema de visão artificial a consumir 1.17A (14,04W/12V)

Figura 4.18: Comparação de imagens adquiridas para um tempo de exposição da câmara de 1/120 s, a superfície superior da peça obliquamente à câmara e à fonte de iluminação e intensidades de iluminação distintas

Conclui-se esta secção, apresentando na figura 4.19 duas imagens adquiridas nos testes finais do sistema de inspeção automática desenvolvido. A técnica de iluminação utilizada foi a **difusa frontal de campo escuro** com as especificações acima indicadas.



(a) Imagem de uma peça a ser inspecionada na superfície superior

(b) Imagem de uma peça a ser inspecionada na superfície lateral

Figura 4.19

4.5 Algoritmo de deteção de defeitos

Este algoritmo tinha como principal objetivo a identificação dos defeitos na superfície da peça.

O pré-processamento através da aplicação de um filtro gaussiano, foi indispensável para suavizar as imagens adquiridas. O filtro gaussiano é um filtro capaz de reduzir o nível de ruído de uma imagem, provocando a suavização da última. O grau de suavização deste filtro depende do valor de um sigma e da dimensão do núcleo utilizado. Posteriormente, segmentou-se a imagem a partir de várias abordagens e compararam-se os resultados obtidos, como será visto de seguida. Por último, com base nas imagens segmentadas, identificaram-se os defeitos nas superfícies da peça.

De seguida, será explicado todo o procedimento adotado, passo a passo.

O código foi desenvolvido na linguagem de programação *C++* utilizando a versão 3.0.0 da biblioteca *OpenCV*, podendo ser consultado na secção C dos anexos.

4.5.1 Segmentação dos defeitos

A segmentação é um processo pelo qual uma imagem é divida em múltiplas regiões, com base num ou mais fatores comuns entre elas, de forma a facilitar a sua análise. Neste caso, essas regiões correspondem às afetadas e não afetadas com defeitos.

Nas secções anteriores, pôde observar-se que, de forma geral, os defeitos a serem detetados estão relacionados com pontos isolados ou em sequência. Por definição, a deteção de pontos deve ser baseada em operadores de 2ª derivada [5], como o Laplaciano. Posto isto, a primeira abordagem tida em conta foi uma baseada no último operador, no entanto, observaram-se melhores resultados numa abordagem baseada no gradiente morfológico, como se verá de seguida.

Também foram testadas abordagens baseadas noutros filtros, que normalmente são usados na deteção de orlas, como o filtro de Sobel e o de Sharr (operadores de 1ª derivada). Contudo, não foram obtidos bons resultados com estas abordagens.

4.5.1.1 Baseada no operador Laplaciano

As figuras 4.21 e 4.22, ilustram os resultados obtidos através da aplicação direta do operador Laplaciano à imagem da figura 4.19a, recorrendo a vários núcleos. Os últimos considerados, foram os correspondentes ao filtro de Sobel (para calcular as derivadas de 1ª e 2ª ordem) e o apresentado na figura seguinte.

$$\begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}$$

Figura 4.20: Um dos núcleos usados pelo operador Laplaciano

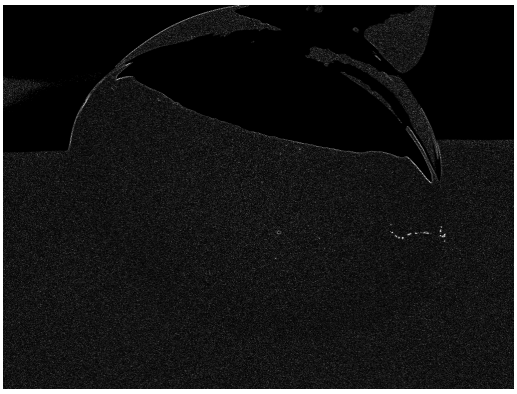
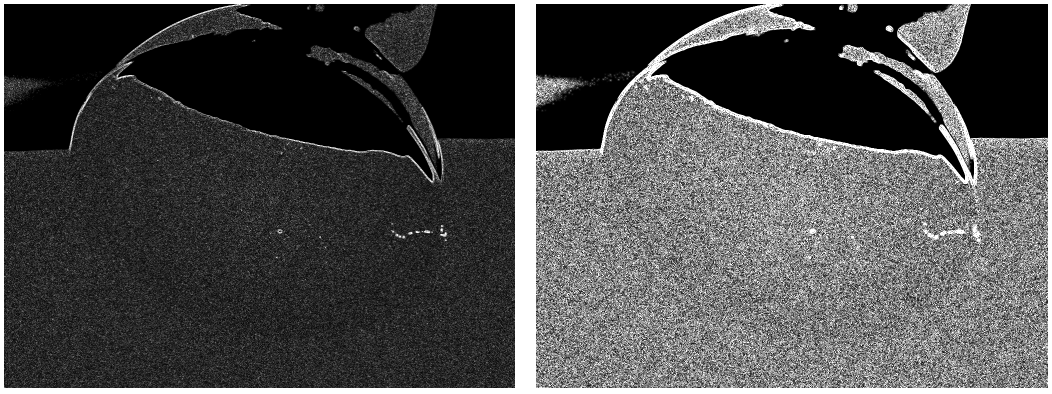
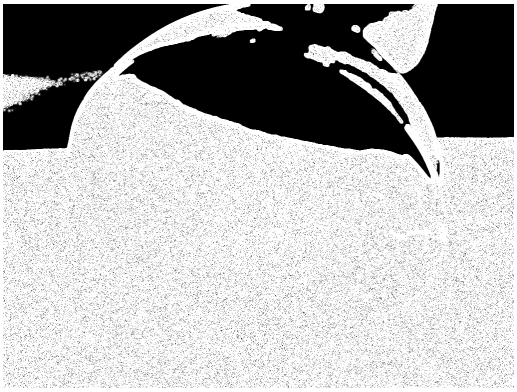


Figura 4.21: Resultado obtido através da aplicação direta do operador Laplaciano, usando o núcleo da figura 4.20



(a) 3x3 (b) 5x5



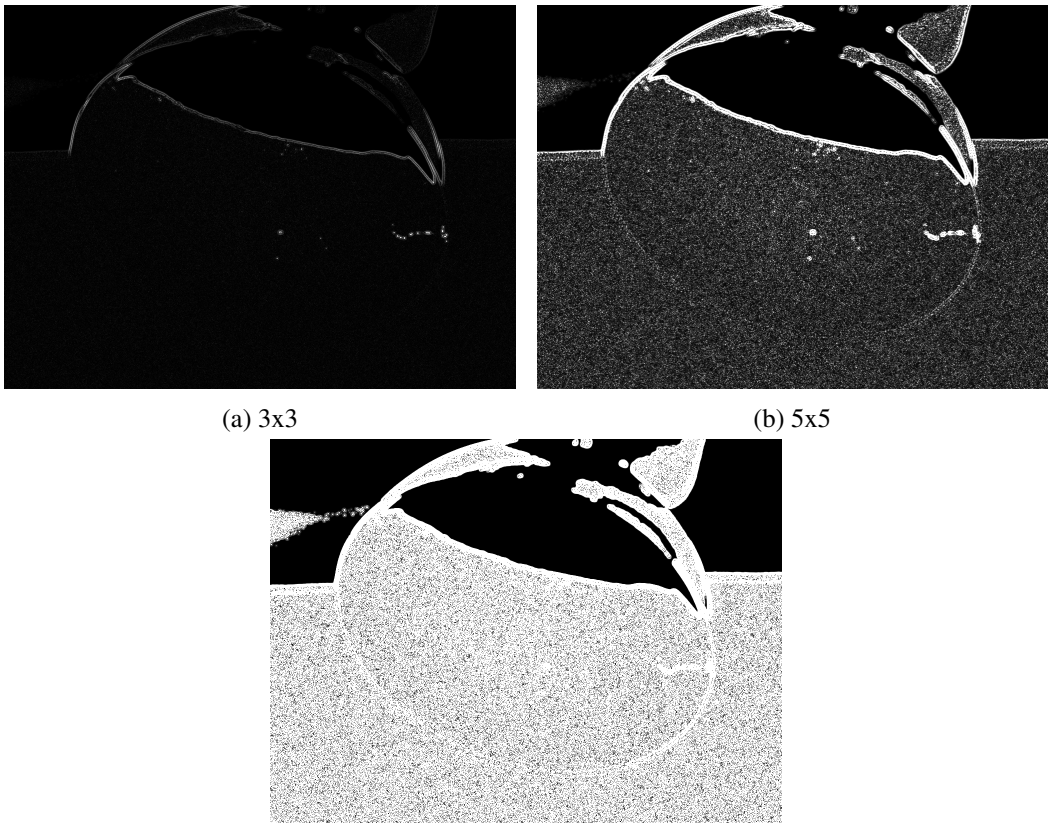
(c) 7x7

Figura 4.22: Resultados obtidos através da aplicação direta do operador Laplaciano, usando núcleos do filtro de Sobel com várias dimensões

Através das imagens acima, pôde verificar-se que era necessária a aplicação prévia de um filtro de suavização, visto que o operador Laplaciano era extremamente sensível ao ruído gaussiano. O filtro escolhido foi o gaussiano, tendo-se obtido uma melhoria clara dos resultados, como se pode observar nas figuras 4.23 e 4.24. A dimensão do núcleo do filtro gaussiano foi ajustada até serem obtidos os melhores resultados, que corresponderam à dimensão 5x5, para a superfície superior

da peça. Também se verificou que, para a última superfície referida, os melhores resultados foram obtidos usando o núcleo apresentado na figura 4.20, na aplicação do operador Laplaciano.

Figura 4.23: Resultado obtido aplicando um filtro gaussiano com núcleo de dimensões 5x5 e posteriormente o operador Laplaciano, usando o núcleo da figura 4.20



(a) 3x3 (b) 5x5

(c) 7x7

Figura 4.24: Resultados obtidos aplicando um filtro gaussiano com núcleo de dimensões 5x5 e posteriormente o operador Laplaciano, usando núcleos do filtro de Sobel com várias dimensões

Posteriormente, binarizou-se a imagem através da aplicação de um limiar. Este foi obtido, novamente, através de várias tentativas até se obter o melhor resultado. O valor escolhido foi 20

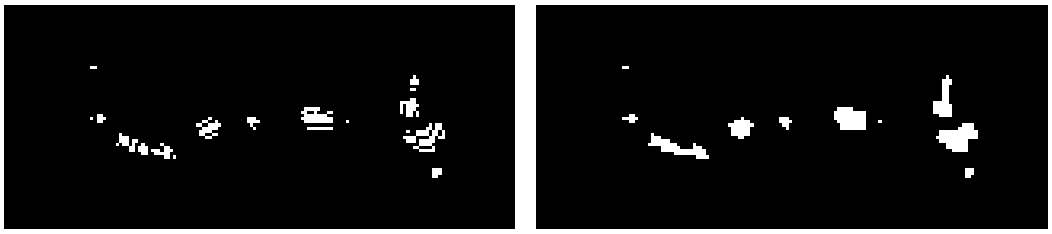
(de 0 a 255, uma vez que a imagem estava na escala de cinzentos), considerando desta forma que, todos pixeis da imagem com valor igual ou superior a este, correspondiam a zonas afetadas com defeito. Desta forma, conseguiu-se obter uma imagem dividida em duas regiões distintas, correspondendo as zonas cujos pixeis têm o valor 1, a regiões de defeito, e as que os pixeis têm o valor 0, a regiões sem defeito (figura 4.25).

Figura 4.25: Resultado da binarização da imagem da figura 4.23 com um limiar de 20

Seguidamente, para fechar pequenos buracos e falhas nas regiões defeituosas, aplicou-se a operação de fecho através do elemento estruturante apresentado imediatamente abaixo. Na figura 4.27, pode observar-se o resultado obtido numa dessas regiões, após ter sido aplicada a operação referida.

$$\begin{bmatrix} 1 & 1 & 1 \\ 1 & \mathbf{1} & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

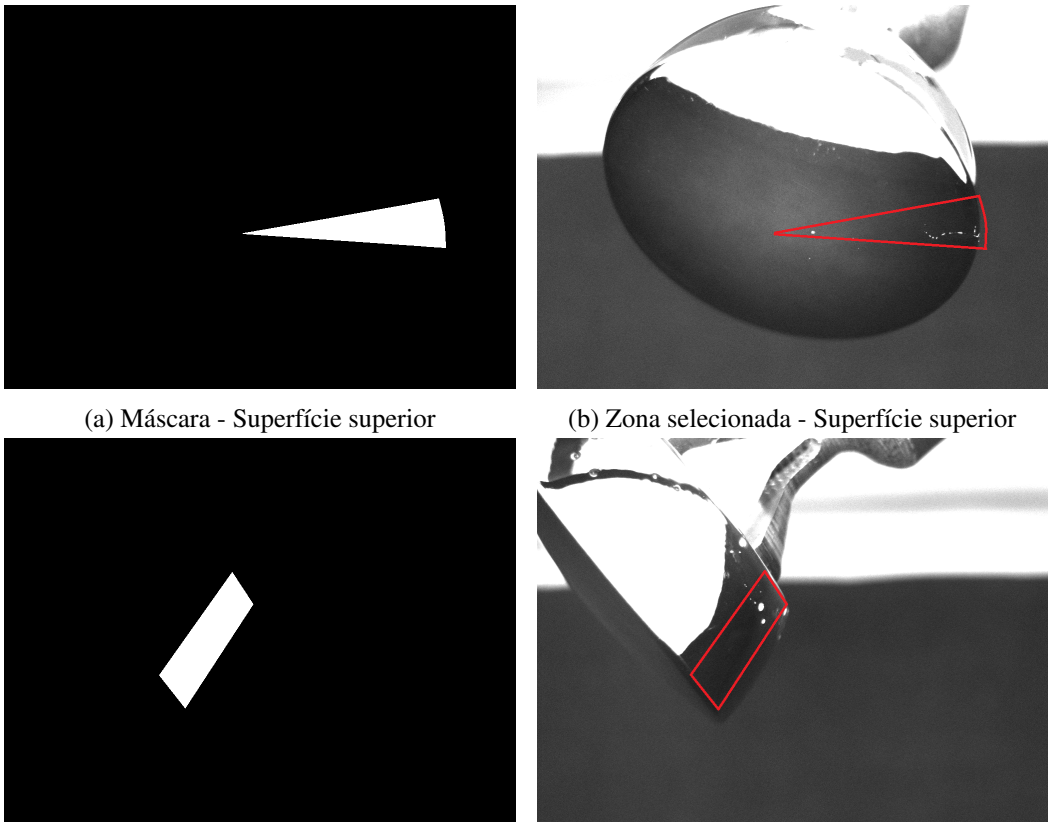
Figura 4.26: Elemento estruturante com origem no elemento a negrito



(a) Antes (b) Depois

Figura 4.27: Comparação de uma zona afetada com defeitos, antes e depois de se aplicar a operação de fecho à imagem

Por último, restou selecionar na imagem a zona que se considerou de interesse, ou seja, a zona que se pretendia inspecionar. Uma vez que a superfície total da peça foi varrida com o auxílio de trajetórias programadas no UR5 e visto que a superfície superior da peça era circular, usou-se uma máscara binária com forma aproximadamente igual à de uma "fatia de um queijo", para se fazer esta seleção. Para a superfície lateral, uma vez que esticada podia ser equiparada a um retângulo, usou-se a forma de um quadrilátero. As imagens da figura 4.28, apresentam as máscaras referidas. A operação utilizada para selecionar a zona de interesse a partir destas máscaras foi o *bitwise AND*. Na figura 4.29, é apresentado o resultado obtido após ser aplicada esta operação sobre a imagem da figura 4.25.



(a) Máscara - Superfície superior

(b) Zona selecionada - Superfície superior

(c) zona selecionada - Superfície lateral

(d) Zona selecionada - Superfície lateral

Figura 4.28: Máscaras aplicadas para selecionar as zonas de interesse, nas duas superfícies consideradas da peça

Figura 4.29: Resultado final da segmentação da superfície superior da peça usando o operador Laplaciano, após ser selecionada a zona de interesse

De forma idêntica, para a superfície lateral, observou-se que o algoritmo apresentado se comportava melhor para a dimensão 7x7 do núcleo do filtro gaussiano e a dimensão 3x3 do núcleo do filtro de Sobel, usado pelo operador Laplaciano (figura 4.30).

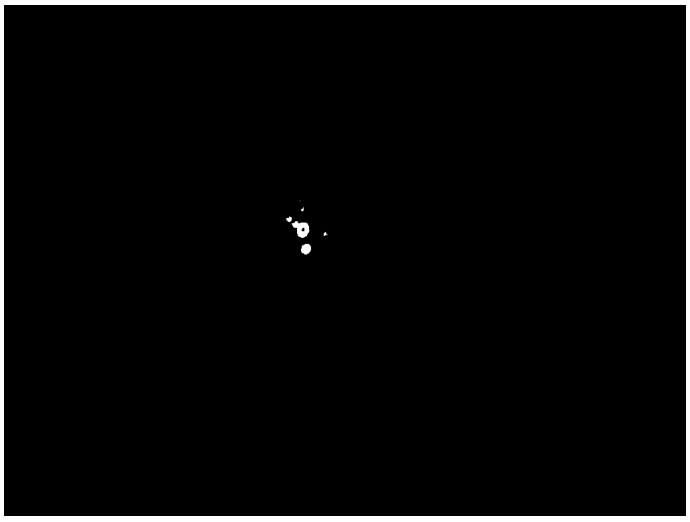


Figura 4.30: Resultado final da segmentação da superfície lateral da peça, usando o operador Laplaciano

O diagrama de atividade apresentado de seguida, resume o algoritmo implementado para segmentar os defeitos, usando o operador Laplaciano. O código implementado está disponível na secção C.1 dos anexos.

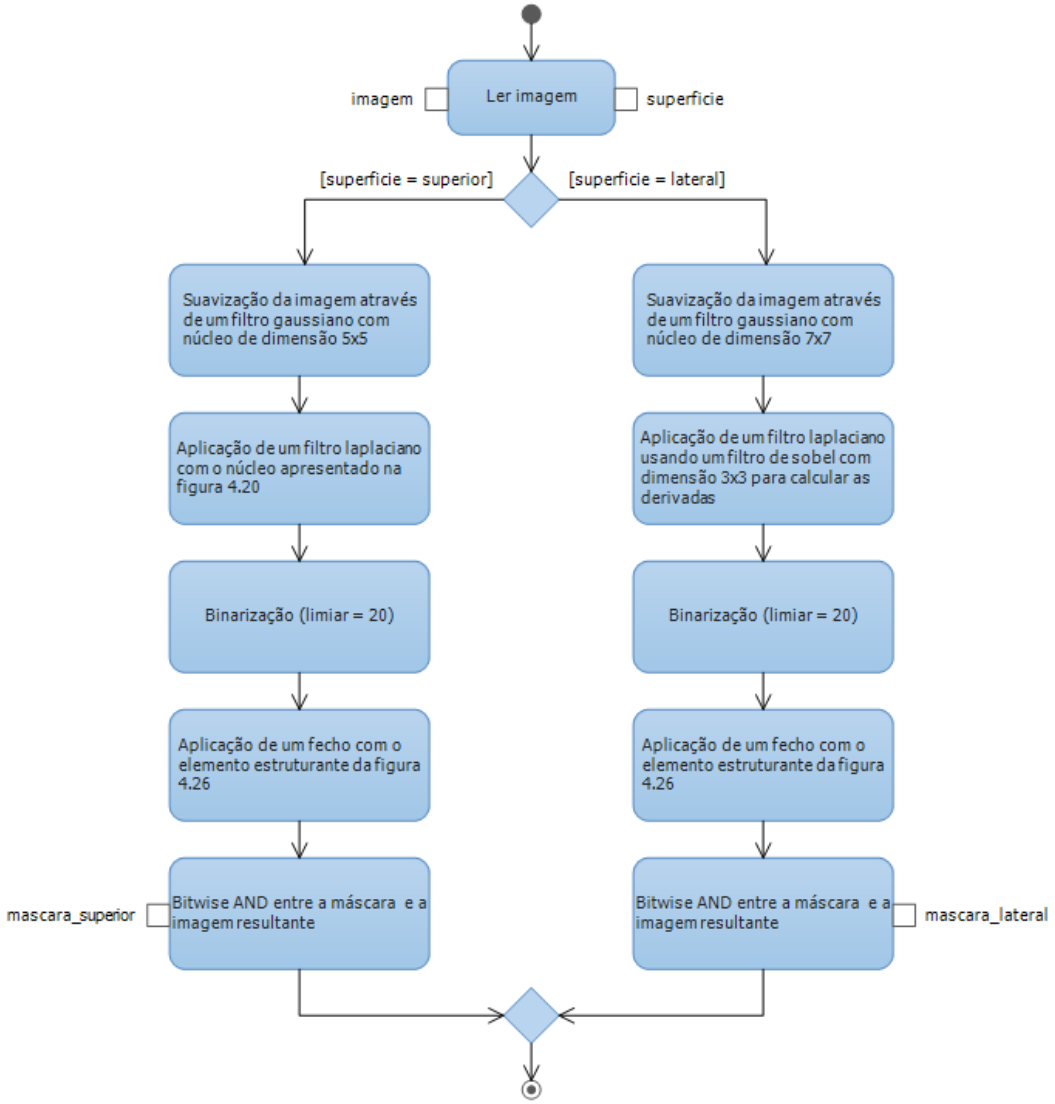


Figura 4.31: Diagrama de atividade do algoritmo de segmentação dos defeitos baseado na aplicação do operador Laplaciano

4.5.1.2 Baseada no gradiente morfológico

O gradiente morfológico é normalmente usado na deteção de orlas. Este operador corresponde à diferença entre a dilatação e a erosão de uma imagem.

Através da mesma estratégia adotada para a segmentação baseada no operador Laplaciano, obtiveram-se os resultados apresentados nas imagens seguintes, para a segmentação baseada no gradiente morfológico. O elemento estruturante considerado na aplicação deste operador, foi o apresentado anteriormente na figura 4.26.

(a) Superfície superior (b) Superfície lateral

Figura 4.32: Resultado final da segmentação das superfícies da peça baseada no gradiente morfológico

O diagrama de atividade apresentado de seguida, resume o algoritmo implementado para segmentar os defeitos usando o gradiente morfológico. O código implementado está disponível na secção C.2 dos anexos.

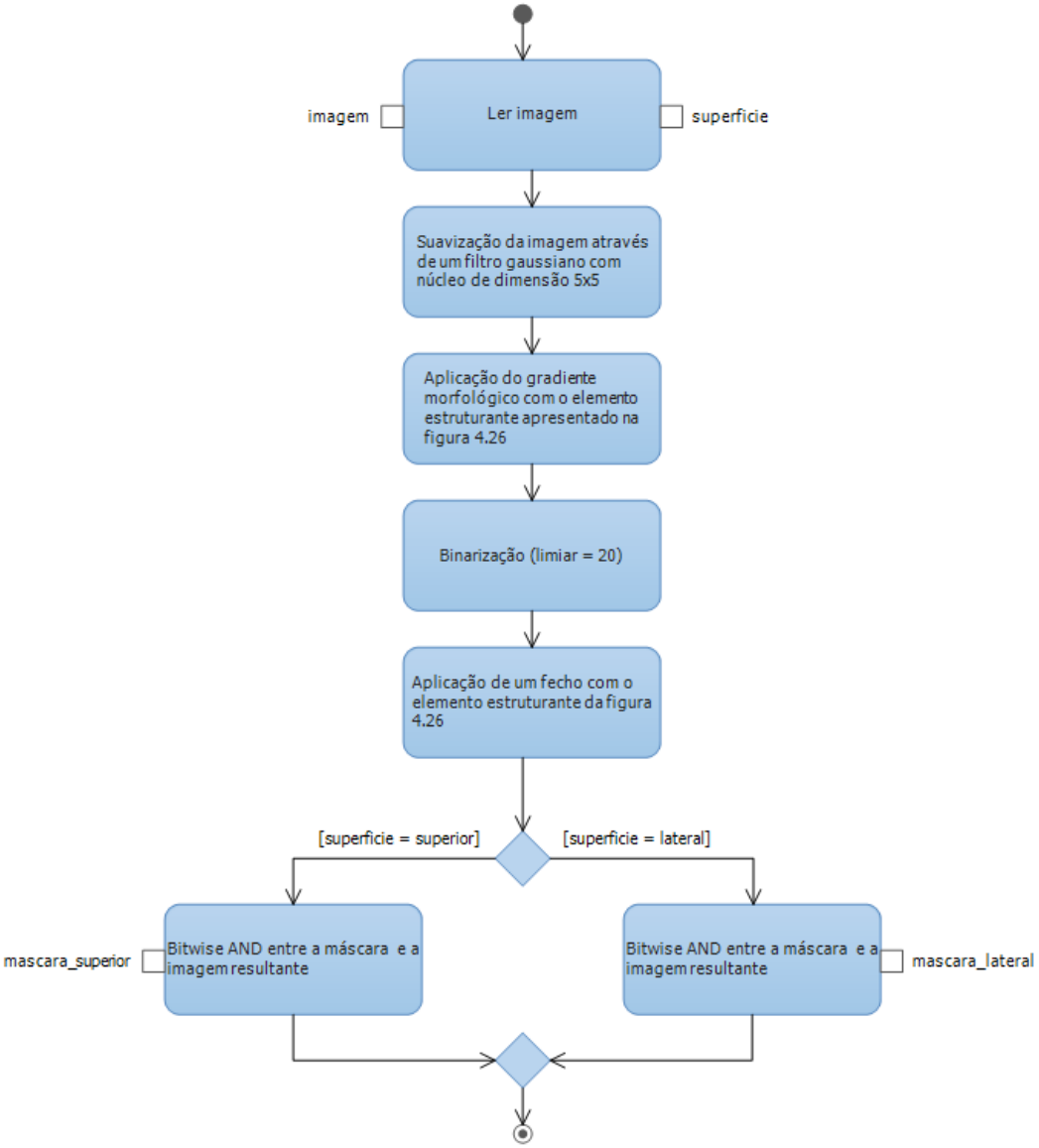
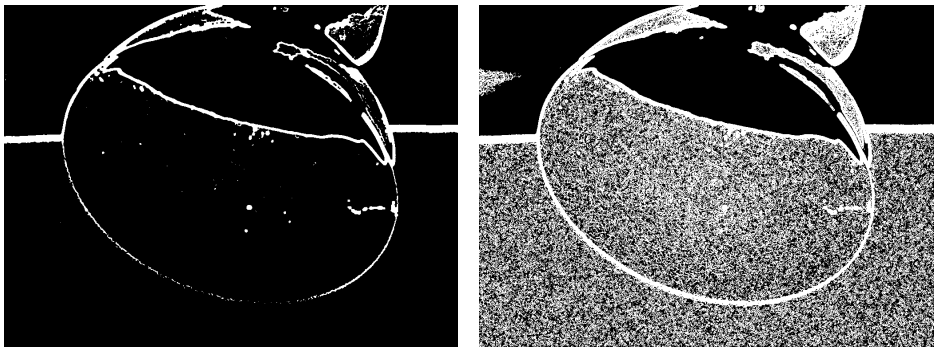


Figura 4.33: Diagrama de atividade do algoritmo de segmentação dos defeitos baseado na aplicação do gradiente morfológico

4.5.1.3 Baseada noutros filtros

Usando estratégias idênticas às apresentadas até agora, mas baseadas em operadores de 1ª derivada, obtiveram-se os resultados da figura seguinte. Os operadores utilizados foram os filtros de Sobel e de Sharr, ambos com núcleos de dimensões 3x3.



(a) Filtro de Sobel (b) Filtro de Sharr

Figura 4.34: Resultado final da segmentação baseada noutros filtros

4.5.1.4 Conclusões

A partir das últimas imagens apresentadas, pôde concluir-se que abordagens baseadas nos filtros de Sobel e de Sharr, não seriam viáveis.

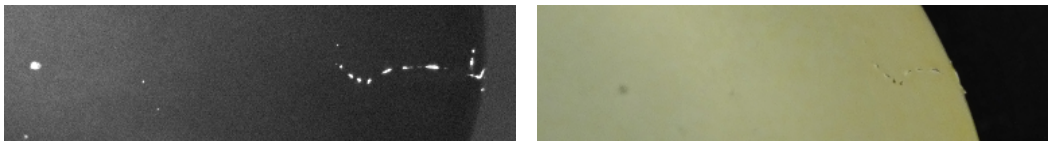
Relativamente à aplicação do filtro de Sobel, pode observar-se a segmentação do contorno da superfície superior da peça, que não corresponde a um defeito. Esta abordagem poderia ser considerada caso se tivesse a certeza que a peça se posicionaria sempre no mesmo local e, desta forma, usava-se uma máscara de seleção mais estreita, para selecionar apenas a zona correspondente à superfície interior da peça. Como isto não acontecia, tal abordagem não pôde ser considerada. No decorrer dos testes realizados, verificou-se que a peça ao rodar na zona de inspeção sofria uma pequena oscilação, devido ao facto da pega de encaixe na peça não ser exatamente concêntrica com a rotação.

As abordagens baseadas no operador Laplaciano e no gradiente morfológico, apresentaram bons resultados. Nas figuras abaixo, pode comparar-se os defeitos segmentados, através destas duas abordagens, e os visíveis nas imagens adquiridas com a câmara usada e com a policromática de um *smartphone*. Foram considerados todos os defeitos que tipicamente aparecem nestas peças. Pôde constatar-se que:

- Os defeitos segmentados através da abordagem baseada no operador Laplaciano, seguem com maior rigor a forma e o tamanho dos defeitos reais, comparativamente aos segmentados através da abordagem baseada no gradiente morfológico (figura 4.35);

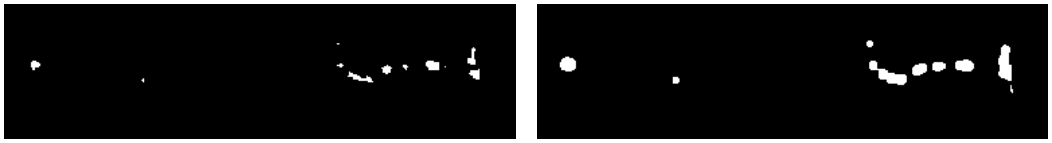
- A abordagem baseada no gradiente morfológico tem uma resposta mais satisfatória para o tipo de defeitos apresentados nas figuras 4.37 e 4.41.

Posto isto, optou-se pela abordagem baseada no gradiente morfológico, uma vez que existe maior garantia na deteção do último tipo de defeito referido.



(a) Defeitos adquiridos com a câmara monocromática usada

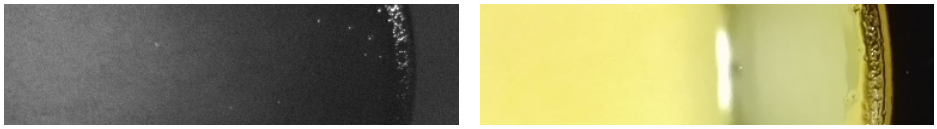
(b) Defeitos adquiridos com a câmara policromática de um *smartphone*



(c) Resultado da segmentação baseada no operador Laplaciano

(d) Resultado da segmentação baseada no gradiente morfológico

Figura 4.35: Comparação entre os defeitos reais presentes na superfície superior da peça e os segmentados - Peça 33 - Ângulo: 120°



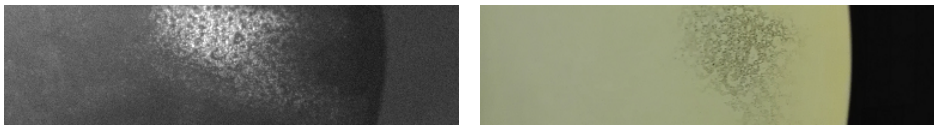
(a) Defeitos adquiridos com a câmara monocromática usada

(b) Defeitos adquiridos com a câmara policromática de um *smartphone*

(c) Resultado da segmentação baseada no operador Laplaciano

(d) Resultado da segmentação baseada no gradiente morfológico

Figura 4.36: Comparação entre os defeitos reais presentes na superfície superior da peça e os segmentados - Peça 31 - Ângulo: 0°



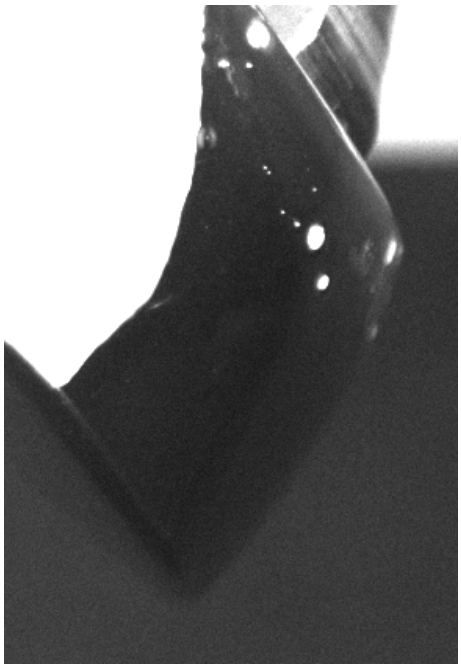
(a) Defeitos adquiridos com a câmara monocromática usada

(b) Defeitos adquiridos com a câmara policromática de um *smartphone*

(c) Resultado da segmentação baseada no operador Laplaciano

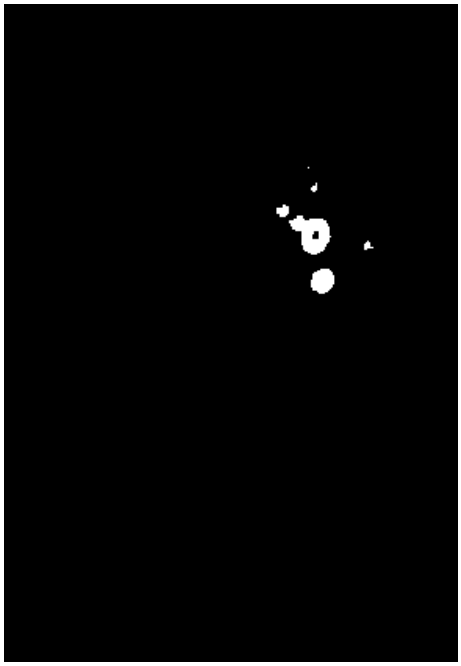
(d) Resultado da segmentação baseada no gradiente morfológico

Figura 4.37: Comparação entre os defeitos reais presentes na superfície superior da peça e os segmentados - Peça 40 - Ângulo: 270°

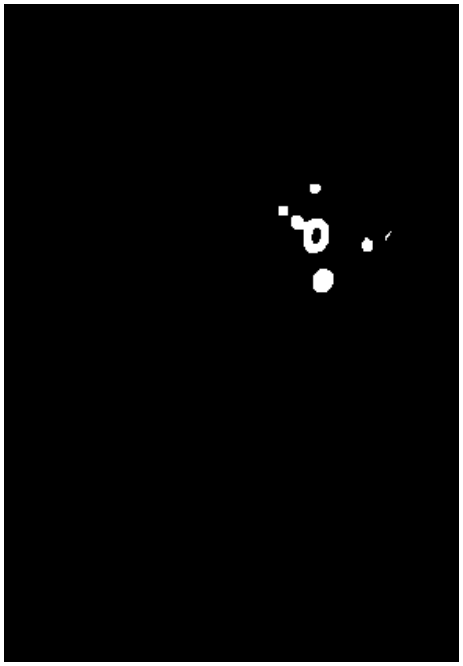


(a) Defeitos adquiridos com a câmara monocromática usada

(b) Defeitos adquiridos com a câmara policromática de um *smartphone*

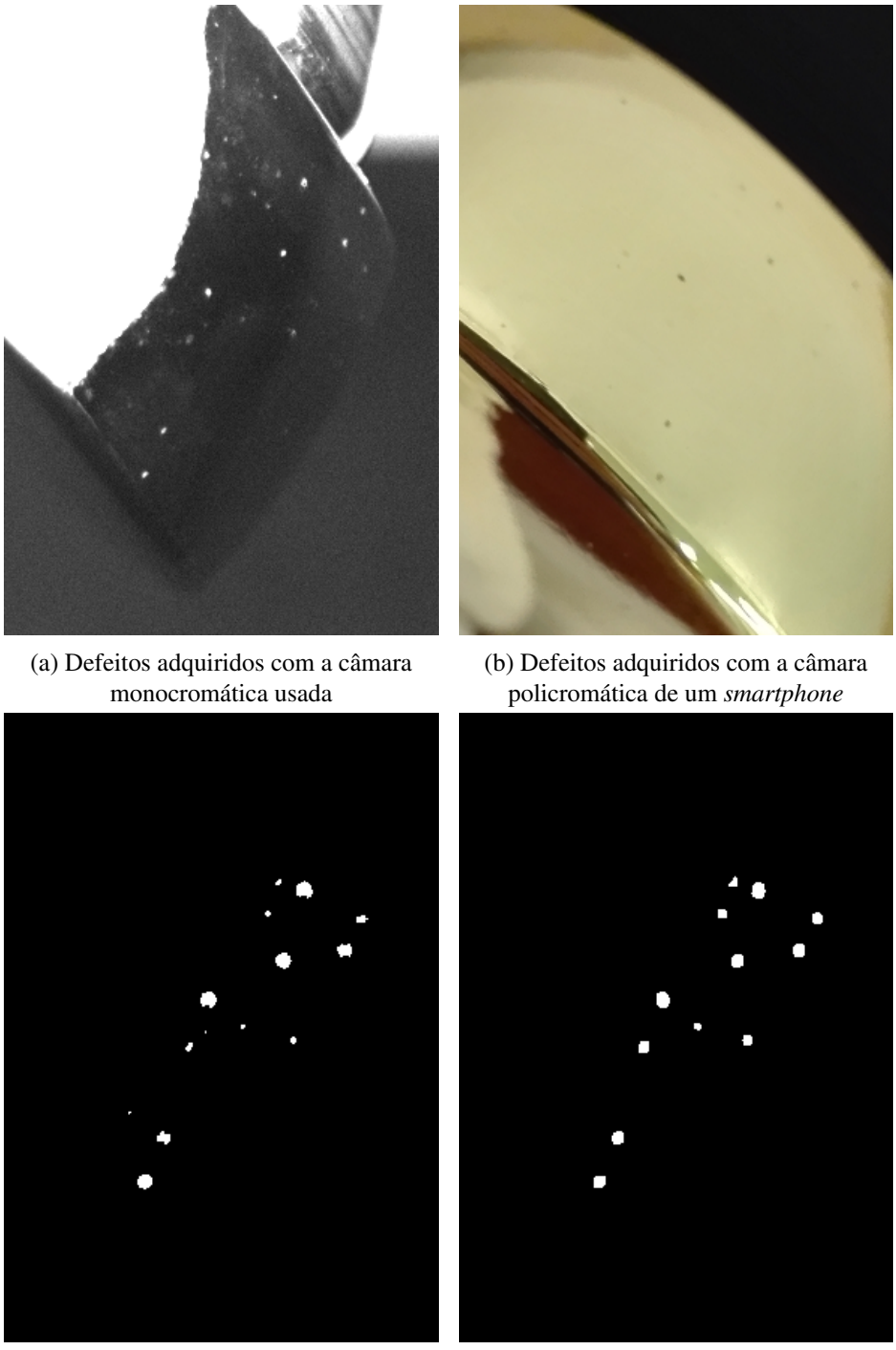


(c) Resultado da segmentação baseada no operador Laplaciano



(d) Resultado da segmentação baseada no gradiente morfológico

Figura 4.38: Comparação entre os defeitos reais presentes na superfície lateral da peça e os segmentados - Peça 32 - Ângulo: 342°



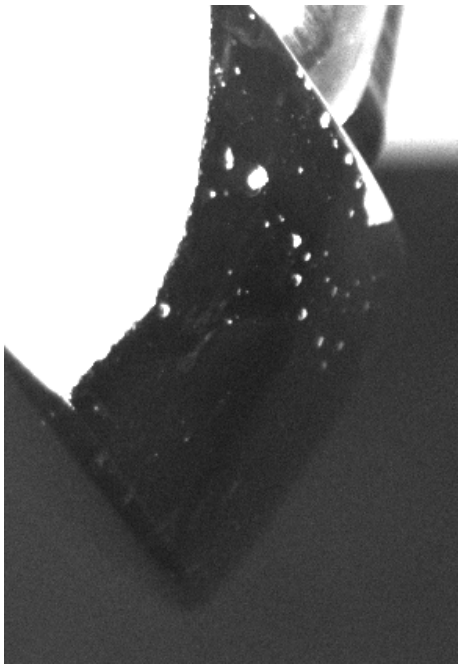
(a) Defeitos adquiridos com a câmara monocromática usada

(b) Defeitos adquiridos com a câmara policromática de um *smartphone*

(c) Resultado da segmentação baseada no operador Laplaciano

(d) Resultado da segmentação baseada no gradiente morfológico

Figura 4.39: Comparação entre os defeitos reais presentes na superfície lateral da peça e os segmentados - Peça 00 - Ângulo: 18°



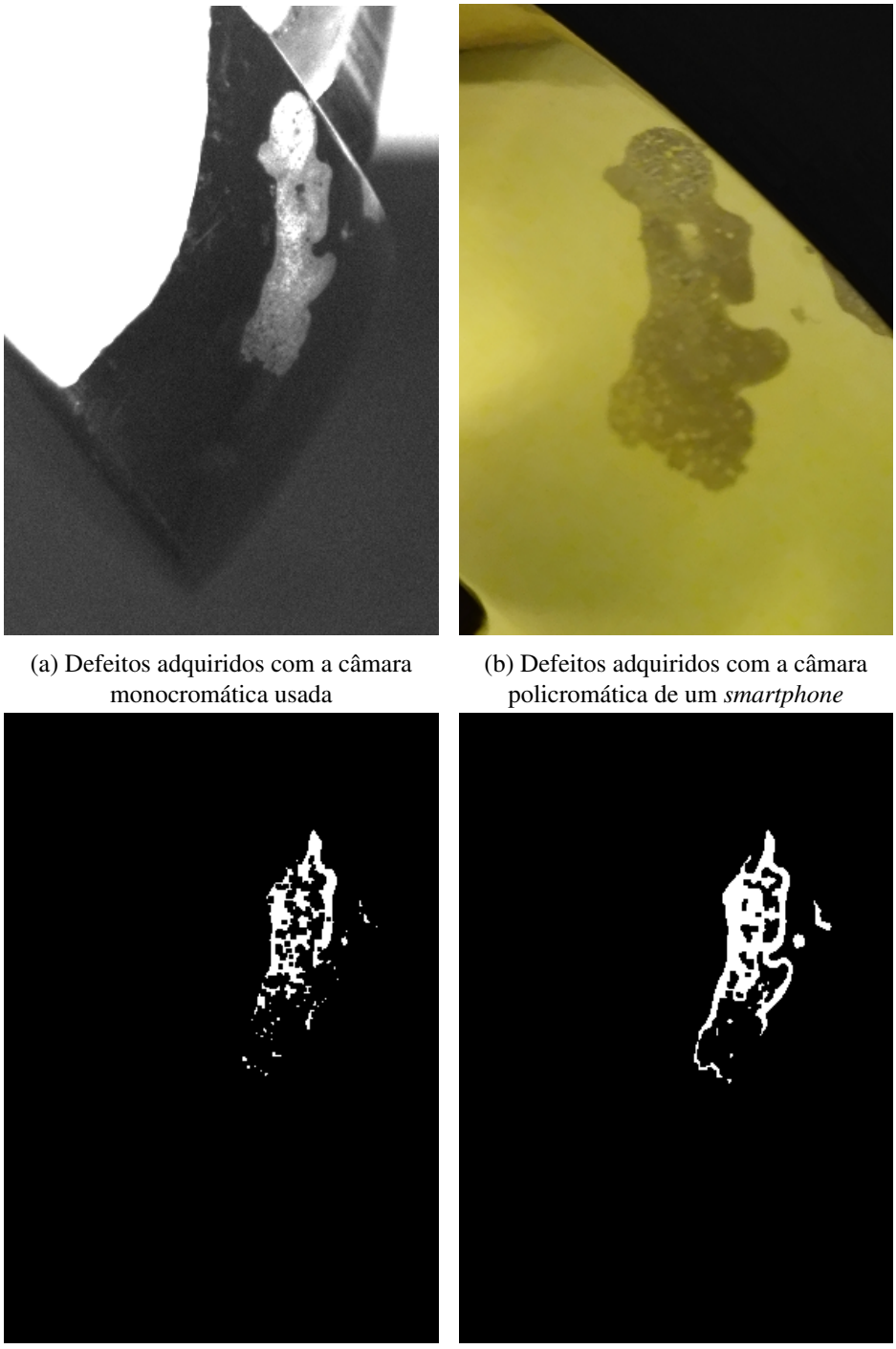
(a) Defeitos adquiridos com a câmara monocromática usada

(b) Defeitos adquiridos com a câmara policromática de um *smartphone*

(c) Resultado da segmentação baseada no operador Laplaciano

(d) Resultado da segmentação baseada no gradiente morfológico

Figura 4.40: Comparação entre os defeitos reais presentes na superfície lateral da peça e os segmentados - Peça 11 - Ângulo: 0°



(a) Defeitos adquiridos com a câmara monocromática usada

(b) Defeitos adquiridos com a câmara policromática de um *smartphone*

(c) Resultado da segmentação baseada no operador Laplaciano

(d) Resultado da segmentação baseada no gradiente morfológico

Figura 4.41: Comparação entre os defeitos reais presentes na superfície lateral da peça e os segmentados - Peça 40 - Ângulo: 18°

4.5.2 Identificação dos defeitos

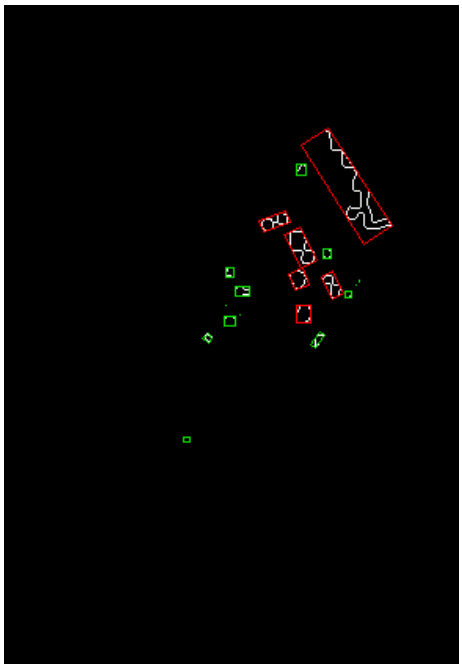
A identificação dos defeitos foi feita com base na imagem previamente segmentada, usando o gradiente morfológico.

Aplicou-se uma função disponibilizada pelo *OpenCV*, para encontrar os contornos dos defeitos segmentados (*findContours*). Através destes contornos e recorrendo a outra função disponibilizada (*moments*), extraíram-se os centros de massa e as áreas dos defeitos segmentados.

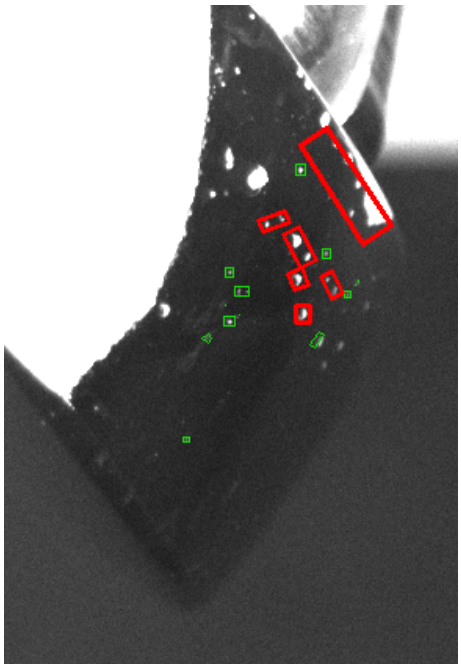
A partir dos valores das áreas dos defeitos, rejeitaram-se aqueles que tinham uma área inferior a 60 px, para a superfície superior, e inferior a 70 px, para a superfície lateral. Estes valores foram obtidos através de comparações entre os defeitos visíveis a olho nu e a sua área correspondente obtida. Em trabalhos futuros, poderá ser desenvolvida uma interface gráfica, onde estes parâmetros são facilmente ajustáveis pelo utilizador.

Utilizando as funções de desenho da mesma biblioteca, desenharam-se retângulos vermelhos em torno dos defeitos aprovados, e verdes em torno dos rejeitados.

Nas figuras seguintes, é apresentado um caso de identificação dos defeitos para cada superfície.



(a) Contornos detetados e respetiva identificação dos defeitos



(b) Imagem adquirida e respetiva identificação dos defeitos

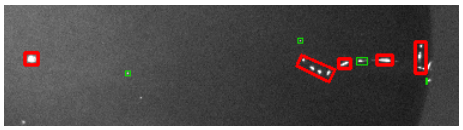
```
6 aproved defects)

Rejected * Defect[0] - Area (M_00) = 12.00 - Perimeter: 14.00 - Mass center(x, y): (366.00, 466.50)
Rejected * Defect[1] - Area (M_00) = 22.00 - Perimeter: 23.31 - Mass center(x, y): (450.03, 402.98)
Rejected * Defect[2] - Area (M_00) = 13.50 - Perimeter: 15.90 - Mass center(x, y): (379.58, 401.49)
Rejected * Defect[3] - Area (M_00) = 37.50 - Perimeter: 23.07 - Mass center(x, y): (393.46, 391.16)
Rejected * Defect[4] - Area (M_00) = 0.00 - Perimeter: 0.00 - Mass center(x, y): (-nan(ind), -nan(ind))
Aproved  * Defect[5] - Area (M_00) = 81.50 - Perimeter: 34.73 - Mass center(x, y): (440.45, 386.63)
Rejected * Defect[6] - Area (M_00) = 0.00 - Perimeter: 0.00 - Mass center(x, y): (-nan(ind), -nan(ind))
Rejected * Defect[7] - Area (M_00) = 13.00 - Perimeter: 13.66 - Mass center(x, y): (468.79, 373.92)
Rejected * Defect[8] - Area (M_00) = 39.00 - Perimeter: 26.49 - Mass center(x, y): (400.99, 372.22)
Rejected * Defect[9] - Area (M_00) = 0.00 - Perimeter: 0.00 - Mass center(x, y): (-nan(ind), -nan(ind))
Rejected * Defect[10] - Area (M_00) = 0.00 - Perimeter: 2.00 - Mass center(x, y): (-nan(ind), -nan(ind))
Aproved  * Defect[11] - Area (M_00) = 70.50 - Perimeter: 44.73 - Mass center(x, y): (458.82, 368.80)
Aproved  * Defect[12] - Area (M_00) = 81.50 - Perimeter: 35.56 - Mass center(x, y): (436.86, 364.24)
Rejected * Defect[13] - Area (M_00) = 22.00 - Perimeter: 18.49 - Mass center(x, y): (393.83, 359.82)
Rejected * Defect[14] - Area (M_00) = 24.50 - Perimeter: 19.07 - Mass center(x, y): (455.31, 347.84)
Aproved  * Defect[15] - Area (M_00) = 164.50 - Perimeter: 66.38 - Mass center(x, y): (438.90, 343.46)
Aproved  * Defect[16] - Area (M_00) = 80.50 - Perimeter: 52.38 - Mass center(x, y): (421.93, 327.67)
Rejected * Defect[17] - Area (M_00) = 30.00 - Perimeter: 22.49 - Mass center(x, y): (439.42, 295.18)
Aproved  * Defect[18] - Area (M_00) = 374.00 - Perimeter: 213.68 - Mass center(x, y): (475.67, 309.79)
```

(c) Informações sobre os defeitos extraídas

Figura 4.42: Resultados obtidos na identificação dos defeitos, numa superfície lateral - Peça 11 - Ângulo: 0°

(a) Contornos detetados e respetiva identificação dos defeitos



(b) Imagem adquirida e respetiva identificação dos defeitos

```
5 aproved defects)

Rejected * Defect[0] - Area (M_00) = 2.00 - Perimeter: 10.83 - Mass center(x, y): (882.50, 473.50)
Rejected * Defect[1] - Area (M_00) = 15.00 - Perimeter: 14.83 - Mass center(x, y): (632.89, 466.00)
Aproved  * Defect[2] - Area (M_00) = 65.00 - Perimeter: 31.31 - Mass center(x, y): (813.95, 457.96)
Rejected * Defect[3] - Area (M_00) = 45.50 - Perimeter: 27.07 - Mass center(x, y): (828.21, 455.78)
Aproved  * Defect[4] - Area (M_00) = 163.50 - Perimeter: 79.21 - Mass center(x, y): (791.30, 462.35)
Aproved  * Defect[5] - Area (M_00) = 85.50 - Perimeter: 36.73 - Mass center(x, y): (847.57, 455.16)
Aproved  * Defect[6] - Area (M_00) = 89.00 - Perimeter: 36.14 - Mass center(x, y): (552.62, 454.20)
Aproved  * Defect[7] - Area (M_00) = 170.00 - Perimeter: 65.80 - Mass center(x, y): (877.59, 453.85)
Rejected * Defect[8] - Area (M_00) = 14.00 - Perimeter: 13.66 - Mass center(x, y): (777.00, 439.00)
```

(c) Informações sobre os defeitos extraídas

Figura 4.43: Resultados obtidos na identificação dos defeitos, numa superfície superior - Peça 33 - Ângulo: 120°

O diagrama de atividade apresentado de seguida, resume o algoritmo implementado para identificar os defeitos. O código implementado está disponível na secção C.3 dos anexos.

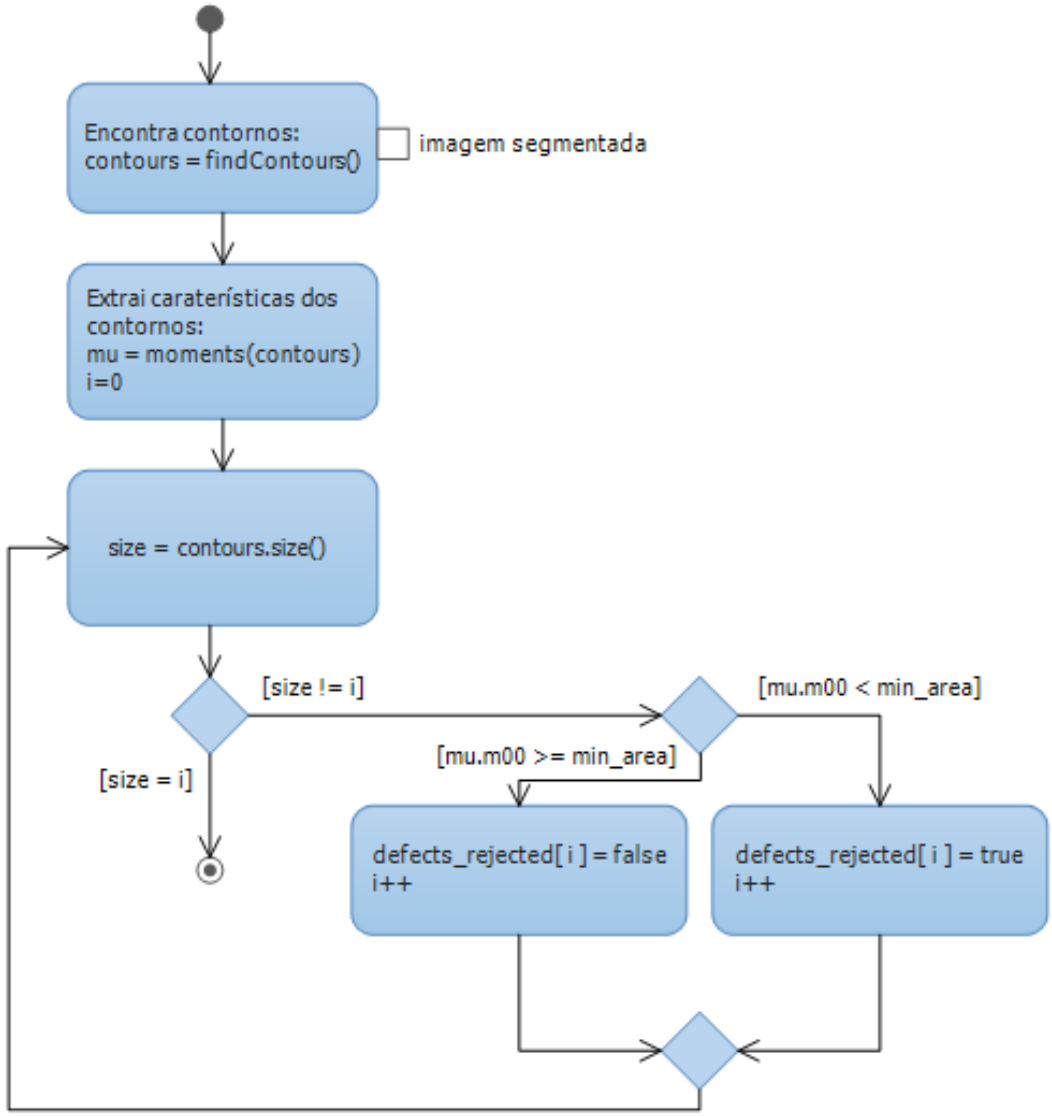


Figura 4.44: Diagrama de atividade do algoritmo de identificação dos defeitos

4.6 Testes e resultados

Os testes finais foram realizados através da integração do algoritmo anteriormente apresentado com trajetórias executadas pelo *UR5*, para que fosse feito o varrimento completo das superfícies das peças.

Desta forma, desenvolveram-se duas aplicações segundo uma arquitetura cliente-servidor (sobre o protocolo TCP/IP), para controlar as trajetórias do manipulador industrial (cliente) e para guardar as imagens adquiridas pela câmara (servidor). Foi decidido não se criar uma aplicação única uma vez que o *software* de inspeção por visão deveria ser o mais genérico possível e desta forma, em trabalhos realizados no futuro, poderá conectar-se a este *software* outro tipo de sistema

de manipulação. O motivo pelo qual se guardaram as imagem foi devido à necessidade de ser feito processamento *offline*, para efeitos de análise de resultados.

Nas seguintes subsecções dos anexos, encontra-se disponível o código desenvolvido para:

- Controlar as trajetórias do *UR5* - C.6;
- Adquirir e guardar as imagens - C.7;
- Processar as imagens adquiridas - C.8.

As tabelas seguintes, apresentam o protocolo de comunicação implementado, pela ordem através da qual as mensagens são enviadas.

Mensagens enviadas pelo cliente	**Significado**
(1) ROBOPOLI_READY_CONFIG _[step_sup]_[step_lat]	Primeira mensagem enviada, após se ter conectado. Informa o servidor que está pronto e envia-lhe o passo de inspeção programado para cada superfície (em graus): step_sup - Passo da superfície superior; step_lat - Passo da superfície lateral.
(2) READY_P_[pos_x][pos_y]	Informa o servidor que uma nova peça está pronta para ser inspecionada e envia-lhe a sua posição no tabuleiro (figura 4.1). pos_x - Posição da peça no eixo x; pos_y - Posição da peça no eixo y.
(3) P_[face]_[total_rotation]	Informa o servidor que a peça está na posição para ser inspecionada uma dada superfície e envia-lhe a sua rotação total nessa posição (em graus). face - Superfície a ser inspecionada ("sup"para superior; "lat"para lateral); total_rotation - Rotação total da peça na posição referida.
(4) P_PARCIAL_END_[face]	Informa o servidor que foi feito o varrimento completo de uma das superfícies.
(5) END_P_[pos_x][pos_y]	Informa o servidor que foi feito o varrimento completo da peça.

Tabela 4.3: Protocolo de comunicação implementado - Mensagens enviadas pelo cliente
Nota: Os parêntesis retos não fazem parte das mensagens

Respostas enviadas pelo servidor	**Significado**
VISIONSYSTEM_READY	Resposta à mensagem nº 1 enviada pelo cliente. O servidor informa que está pronto.
READY_P_[pos_x][pos_y]_OK	Resposta à mensagem nº 2 enviada pelo cliente. O servidor informa que está pronto para inspecionar a peça especificada.
P_[face]_[total_rotation]_OK	Resposta à mensagem nº 3 enviada pelo cliente. O servidor informa que já adquiriu a imagem correspondente à posição especificada.
P_PARCIAL_END_[face]_OK	Resposta à mensagem nº 4 enviada pelo cliente. O servidor informa que tomou conhecimento da finalização do varrimento completo de uma das superfícies.
END_P_[pos_x][pos_y]_OK	Resposta à mensagem nº 5 enviada pelo cliente. O servidor informa que tomou conhecimento da finalização do varrimento completo da peça.

Tabela 4.4: Protocolo de comunicação implementado - Mensagens enviadas pelo servidor
Nota: Os parêntesis retos não fazem parte das mensagens

Desenvolveu-se uma pequena interface com o utilizador na aplicação de controlo do manipulador, para que fossem especificadas as configurações necessárias para a execução das trajetórias e dar início às mesmas (figura 4.45). Estas configurações foram guardadas automaticamente em ficheiros de texto, para que não fossem perdidas entre várias execuções.

O passo utilizado foi 15° para a superfície superior e 18° para a superfície lateral. Estes valores foram obtidos de forma aproximada recorrendo a uma peça de teste com uma marca (figura 4.46), para se ter a perceção da variação angular suportada até que a marca começasse a ficar desfocada, ou a entrar numa zona com reflexão especular. A marca utilizada foi uma linha, para posteriormente auxiliar na criação das máscaras apresentadas na figura 4.28.

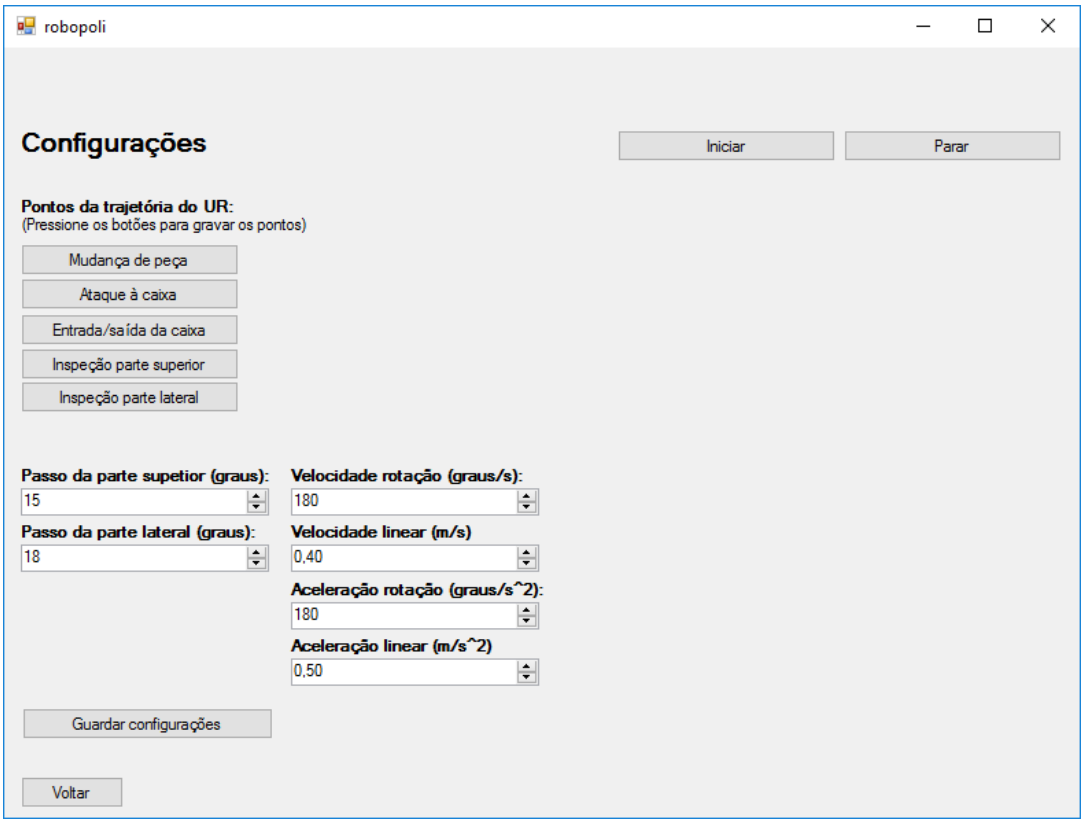
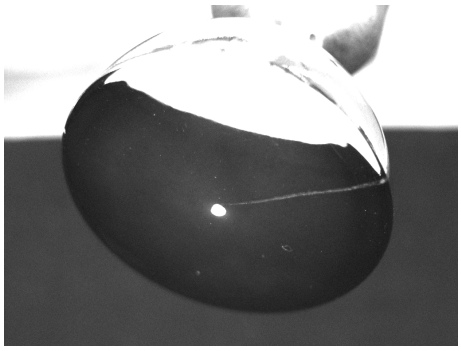
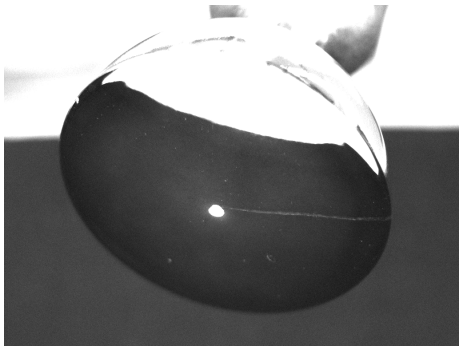


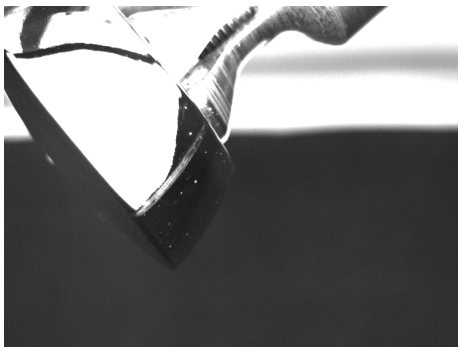
Figura 4.45: Interface desenvolvida para especificar as configurações das trajetórias utilizadas para o varrimento completo das superfícies das peças



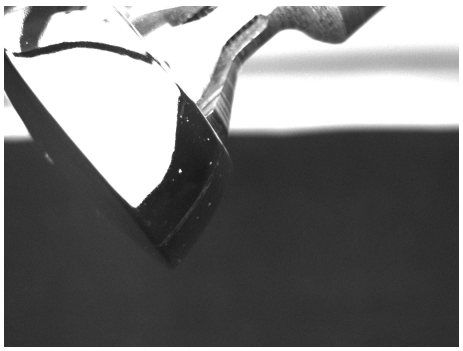
(a) Superfície superior - Ângulo: 0°



(b) Superfície superior - Ângulo: 15°



(c) Superfície lateral - Ângulo: 0°



(d) Superfície lateral - Ângulo: 18°

Figura 4.46: Peça de teste utilizada para o estudo do passo a utilizar na inspeção de cada superfície

Com os passos especificados, para cada peça teve-se 24 (360°/15°) imagens para a superfície superior e 20 (360°/18°) imagens para a superfície lateral, 44 imagens no total.

De forma a serem observados os resultados obtidos, foram reconstruídas as superfícies das peças, a partir das imagens com os defeitos assinalados. Para a superfície superior, a reconstrução consistiu nas três etapas seguintes:

1. Transformação de perspetiva da imagem cortada pela máscara indicada na figura 4.28a, para a posição correspondente aos 0° do círculo correspondente a esta superfície (ver função *transform_sup_zone* na secção C.4 dos anexos). A transformação de perspetiva é calculada com base nas posições de 4 pixeis de partida e de chegada conhecidos;

2. Rotação da imagem obtida no ponto anterior para a sua posição relativa na peça (ver função *rotate* na secção C.4 dos anexos);

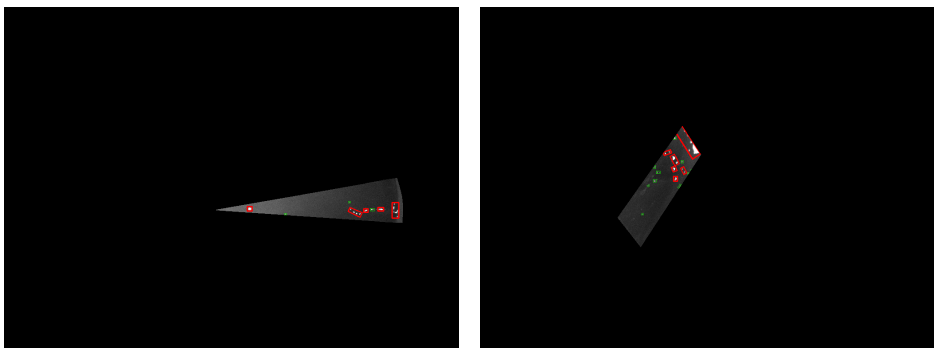
3. Aplicação de um *bitwise OR* entre todas as imagens da peça para esta superfície.

Para a superfície lateral, a reconstrução consistiu nas duas etapas seguintes:

1. Transformação de perspetiva da imagem cortada pela máscara indicada na figura 4.28c, para a sua posição relativa num retângulo representativo desta superfície (ver função *transform_lat_zone* na secção C.5 dos anexos);

2. Aplicação de um *bitwise OR* entre todas as imagens da peça para esta superfície.

Poderá reparar-se nas imagens posteriormente apresentadas nos resultados obtidos, que as reconstruções referidas levaram ao aparecimento de linhas orientadas radialmente, para a superfície superior, e de linhas verticais, para a superfície lateral. Contudo, estas linhas não afetaram a inspeção da peça, uma vez que são uma consequência da junção das imagens, após já terem sido inspecionadas. Na figura seguinte, são apresentadas duas imagens de cada superfície, antes de serem processadas pelo algoritmo de reconstrução.



(a) Superfície superior (b) Superfície lateral

Figura 4.47: Imagens inspecionas, antes de serem processadas pelo algoritmo de reconstrução

Todos os resultados obtidos encontram-se na secção D dos anexos, sendo que a subsecção D.1 corresponde às peças que eram consideradas boas e a D.2 às que eram consideradas más.

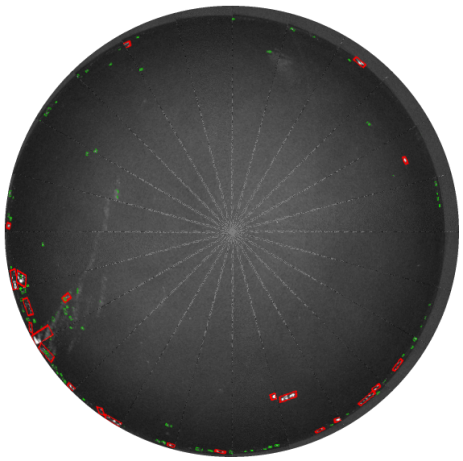
Os tempos máximos e típicos de processamento do algoritmo de deteção de defeitos (carregamento da imagem, segmentação e identificação de defeitos), foram os indicados na tabela 4.5. O computador pessoal utilizado na realização dos testes foi o apresentado na secção 4.1. O tempo de reconstrução das superfícies não foi considerado, uma vez que é um processo utilizado apenas para a observação de resultados. O tempo de varrimento das peças através das trajetórias executadas, também não foi considerado devido ao facto do manipulador parar e esperar que seja adquirida a imagem em todos os passos de inspeção, como sugere o protocolo de comunicação implementado. Este facto fez com que cada peça demorasse cerca de 30 segundos a ser varrida. Optou-se por esta abordagem para que fosse garantido, nesta fase, o sincronismo entre as imagens adquiridas e o ângulo de rotação da peça. Contudo, este processo poderá facilmente ser acelerado, colocando a peça a rodar a uma velocidade constante e adquirindo as imagens entre variações de tempo constantes.

Superfície	**Tempo de processamento (ms)**		**Nº de frames**	**Tempo total máximo (ms)**
	Típico	Máximo		
Superior	150	288	24	6912
Lateral	150	195	20	3900
			Total	10812

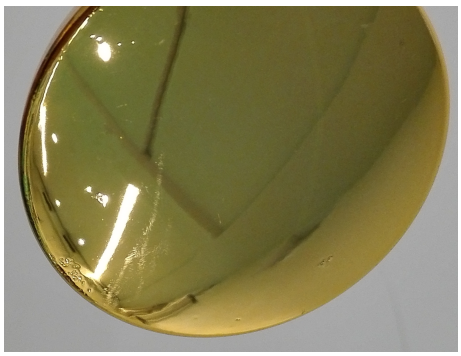
Tabela 4.5: Tempos de processamento obtidos do algoritmo de deteção de defeitos (carregamento da imagem, segmentação e identificação de defeitos)

Relativamente aos resultados obtidos na deteção dos defeitos, pôde concluir que, de forma geral, foram obtidos bons resultados com esta abordagem, uma vez que os defeitos assinalados nas imagens se assemelham na sua maioria aos visíveis a olho nu. Comparando as superfícies superiores das peças boas e más, é bem percetível a diferença na quantidade e no tamanho dos defeitos detetados (exepto o caso da peça 01 (D.2a), que a sua classificação como peça boa deixa algumas dúvidas). Verificou-se também que a presença de defeitos nas superfícies laterais, não tem grande peso na avaliação das peças, caso estes sejam de pequena dimensão. Contudo, identificaram-se algumas limitações na deteção de alguns tipos de defeitos, como por exemplo:

- No caso da peça 01, é observável um "arranhão"que, por ser de certa forma suave e não retirar toda a refletividade e espelhamento nessa zona, acabou por não ser detetado. A figura seguinte ilustra o caso referido. Para solucionar este problema, terão de ser estudados outros operadores usados na deteção de orlas e outras técnicas de condicionamento do ambiente de aquisição.



(a) Defeitos detetados



(b) Imagem a cores
Nota: O defeito só se tornava percetível, quando espelhava nas suas imediações uma fonte de iluminação intensa

Figura 4.48: Peça 01 - Comparação entre os defeitos detetados e os reais

- A inclinação da peça em relação à fonte de iluminação e à câmara, o passo adotado no varrimento da superfície superior e a focagem da lente, deviam ter sido estudados tendo em conta os defeitos na borda da superfície superior. Como se pode observar na figura seguinte, o defeito na borda era contínuo, no entanto, como uma parte dessa zona deixou de ficar tão bem focada, os defeitos não foram detetados dessa forma. Na peça apresentada, este facto não impediria que a peça fosse rejeitada. No entanto, se os defeitos coincidissem apenas com a zona que não é tão bem focada, poderiam passar despercebidos.

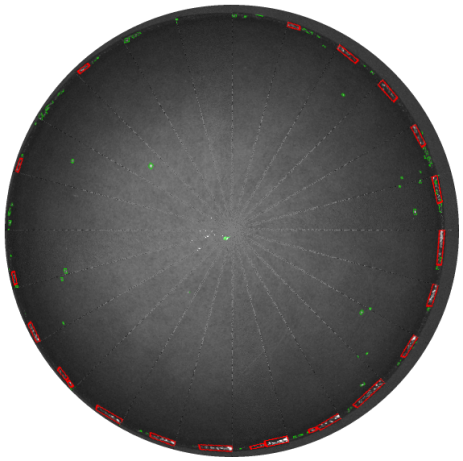


Figura 4.49: Defeitos detetados descontínuos na borda da superfície da superior, quando na realidade eram contínuos

- Por último, devido a variações elevadas nos pixeis dentro de defeitos de grandes dimensões, são detetados sub-defeitos, tendo faltado rejeitar os mesmos. Esta rejeição poderia ser feita com base na posição e na área ocupada pelos defeitos. A figura seguinte apresenta a situação referida.

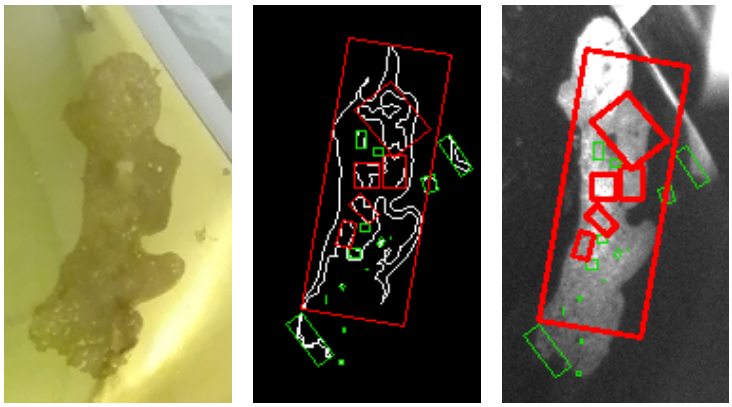


Figura 4.50: Presença de sub-defeitos (assinalados a vermelho dentro do retângulo vermelho maior)

4.7 Conclusões

O objetivo principal desta parte do trabalho foi desenvolver um sistema o mais genérico possível, que permitisse a inspeção de qualquer tipo de peça com superfície altamente refletora e espelhada. Pôde concluir-se que, para a inspeção deste tipo de peças pintadas ou muito bem polidas, o conceito do sistema de inspeção apresentado é viável e necessitará apenas de pequenas adaptações no que diz respeito à posição das peças na zona de inspeção, às trajetórias executadas no varrimento das peças e às máscaras de seleção das zonas de interesse.

O resultado obtido com o condicionamento do ambiente de aquisição, através da técnica de iluminação difusa frontal de campo escuro e do isolamento em relação ao ambiente exterior com caixa apresentada, foi bastante satisfatório pelos motivos de seguida expostos. Os defeitos foram realçados de forma conveniente para, posteriormente, serem detetados. O fundo da caixa revestido com veludo preto, mostrou ser uma solução ótima para cobrir o fundo das imagens. Isto porque, como não se tinha a certeza da posição exata onde a peça iria ser inspecionada, a máscara de seleção teve de ser alargada, acabando por selecionar um bocado do fundo em causa. No entanto, como o fundo escolhido não provoca uma variação elevada nas intensidades dos pixéis de fronteira com a peça, este facto acabou por não ser um problema.

O algoritmo desenvolvido, com a segmentação dos defeitos através da aplicação do gradiente morfológico e, posteriormente, com a deteção dos defeitos através dos contornos dos defeitos segmentados, apresentou bons resultados, uma vez que se obteve uma grande semelhança entre os defeitos detetados e os visíveis a olho nu. Contudo, este algoritmo apresentou certas limitações, como a não deteção de defeitos mais suaves, o problema referido dos defeitos na borda da superfície superior e a não rejeição dos sub-defeitos, como foi demonstrado na secção anterior.

Em trabalhos futuros, será necessário desenvolver um algoritmo de avaliação das peças, para que, com base nas posições, densidades e tamanhos dos defeitos, se determine se uma peça deve ser ou não rejeitada. Também terão de ser estudadas novas soluções, de forma a diminuir o número de imagens adquiridas, visto que a cada peça correspondem 44 imagens e isso fez com que este

sistema levasse cerca de 11 segundos (no pior dos casos) para inspecionar uma peça, o que é demasiado tempo para uma aplicação de inspeção em tempo real. As soluções a estudar para responder ao último problema poderão passar pela introdução de um conjunto de câmaras estrategicamente posicionadas e pelo processamento paralelo das imagens.

Capítulo 5

Conclusões gerais e trabalho futuro

Neste capítulo, serão expostas as conclusões gerais e o trabalho futuro deste projeto.

Relativamente à solução proposta de polimento robotizado programado por demonstração humana, o estudo realizado sobre sistemas de robotizados com controlo de força faziam prever algumas dificuldades neste âmbito. Contudo, este problema seria ultrapassado através da utilização do manipulador industrial *UR5*, que já possuía controlo de força. Contudo, uma vez que este controlo era feito recorrendo às correntes elétricas nos motores das articulações do manipulador, os testes realizados mostraram que o seu desempenho dependia fortemente da posição dessas articulações e, no melhor caso, a força controlada era pouco precisa. Desta forma, no trabalho futuro deste projeto poderá vir a ser testado, se para as peças em causa a polir, um controlo com pouca precisão e sensibilidade é suficiente. Caso não o seja, existem vários sensores de força/binário no mercado com alta repetibilidade e precisão, que podem ser usados pelo *UR5*, para que o controlo em causa seja executado com alta precisão. O sensor utilizado neste projeto para a gravação das forças, pode ser ligado a este manipulador através do protocolo de comunicação *Modbus*, muito utilizado em redes de automação industrial (clique aqui para ler uma demonstração de utilização do sensor de força/binário *Gamma* da *ATI Industrial Automation* com o *UR5*).

A programação do manipulador em causa, foi feita através do envio de instruções *URScript*, como foi dito anteriormente. A comunicação com o manipulador foi realizada com sucesso, contudo, no final, veio a descobrir-se que esta deveria ter sido feita pela porta 30003, e não pela 30002, uma vez que a frequência da comunicação é de 125 Hz, e na utilizada é de apenas 10 Hz. Para o envio de instruções *URScript*, esta alteração é imediata (alterando apenas o número da porta), no entanto, não o é para a leitura do estado interno do manipulador, uma vez que a trama enviada nesta porta tem um formato ligeiramente diferente. Desta forma, terá de ser adaptado o algoritmo de leitura da trama referida. Futuramente, para evitar o envio de programas repetidos para o manipulador, poderá ser implementado um protocolo FTP (*File Transfer Protocol*), que guarda os programas na memória interna do controlador do manipulador. Desta forma, um operador será capaz de dar início à execução dos programas através do *teach pendant*, sem ser necessária a ligação a um PC.

No que diz respeito à demonstração das forças e das trajetórias, conseguiu-se implementar

corretamente uma estratégia para que estas fossem gravadas sincronizadamente e, posteriormente, fossem geradas as instruções a enviar para o manipulador industrial, para que este executasse as trajetórias demonstradas. Também foi testado o envio das instruções de controlo de força (*force_mode* e *end_force_mode*), no entanto, estas instruções não foram geradas a partir das forças demonstradas. Ficou ainda por implementar a gravação num ficheiro de texto ou numa base de dados, dos pontos associados às trajetórias e às forças demonstradas.

Concretamente em relação à *framework 6DMimic*, sugere-se que futuramente esta possa ser melhorada, através do acrescento das seguintes funcionalidades:

- Ser possível configurar o ponto central da ferramenta, em relação ao referencial associado ao centro do marcador luminoso. Com esta melhoria, a lista retornada com pontos da trajetória demonstrada, já estaria associada ao referencial do ponto central da ferramenta a utilizar, o que tornaria menos complexo o processo de calibração dos referenciais apresentado anteriormente;

- O protocolo de comunicação com esta *framework*, poderá fazer o *streaming online* dos pontos das trajetórias registados. Através desta melhoria, não se teriam de utilizar ficheiros de output para transferir os pontos das trajetórias para uma aplicação *Master*, o que tornaria esta operação impercetível para o utilizador;

- Deverá ser acrescentada alguma redundância ao sistema, para se eliminar o problema do marcador oculto. Como o sistema apresentado tem apenas duas câmaras, posicionadas lado a lado, e um marcador luminoso, ao serem demonstradas rotações, este marcador corria o risco de ficar oculto quando se posicionava atrás da ferramenta (em relação ao ponto de vista das câmaras). Esta situação poderá ser resolvida através da utilização de mais do que um marcador, posicionados de forma a que pelo menos um deles esteja sempre visível pelas câmaras, através da inserção de mais câmaras posicionadas de forma diferente, ou então, através da inserção de uma IMU (*Inertial Measurement Unit*) no marcador.

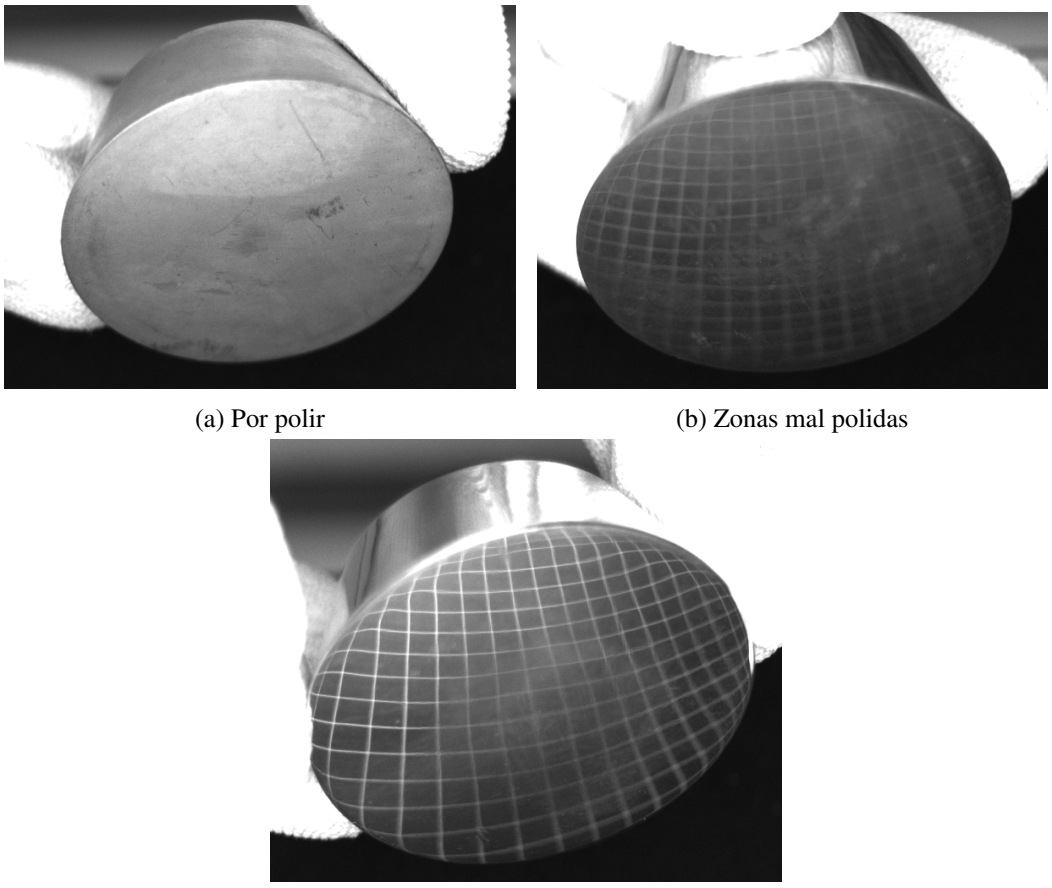
No que diz respeito ao sistema de inspeção automática por visão artificial de peças com superfície altamente refletora e espelhada, a revisão bibliográfica fazia prever alguma dificuldade na aquisição de imagens com qualidade suficiente para serem inspecionadas, devido às reflexões especulares provocadas pela iluminação. Para contornar este problema, desenvolveu-se uma técnica eficaz de condicionamento do ambiente de aquisição das imagens, através da utilização de uma caixa de isolamento com o ambiente exterior e da utilização do tipo de iluminação difusa frontal de campo escuro.

O algoritmo desenvolvido para identificar defeitos presentes neste tipo de peças, apresentou bons resultados, na medida em que os defeitos identificados correspondiam aos visíveis através do olho humano. Futuramente, deverá ser desenvolvido um algoritmo de classificação das peças, com base nas posições e áreas dos defeitos identificados, capaz de rejeitar peças defeituosas. Também terá de ser estudada uma estratégia para diminuir o número total de imagens adquiridas para

inspecionar uma única peça. Isto porque, para as peças de teste consideradas, tinham de ser adquiridas 44 imagens, o que tornava demoroso o seu processamento. No pior dos casos, estimou-se um tempo de processamento total de aproximadamente 11 s. Este problema poderá ser corrigido, estudando soluções que utilizem um conjunto de câmaras estrategicamente posicionadas (e não apenas uma) e que processem as imagens adquiridas paralelamente.

A comparação dos algoritmos de visão baseadas em *software* grátis (licença BSD), como por exemplo a biblioteca do *OpenCV*, com abordagens baseadas em *software* comercial, como por exemplo o *HALCON*, proposta nos objetivos desta dissertação não foi feita, pois como se conseguiu obter bons resultados com o *OpenCV*, sem que para isso tenha sido despendido muito tempo, não faria sentido testar abordagens baseadas em *software* comercial.

Para terminar, nos objetivos deste projeto, também foi proposto o desenvolvimento de um algoritmo capaz de classificar o estado de polimento de peças metálicas. Já não houve tempo suficiente para se abordar este problema, no entanto, pensou-se numa possível solução a implementar, apresentada de seguida. As peças metálicas bem polidas distinguem-se claramente das não polidas através da refletividade e do espelhamento da sua superfície, como foi possível observar na figura 3.11, apresentada anteriormente. Desta forma, pensou-se que o estado de polimento possa vir a ser detetado através do espelhamento de um padrão previamente conhecido, na superfície da peça a inspecionar. Caso esse padrão não fosse detetado, ou as linhas não fossem bem definidas, em certas zonas da superfície, significaria que essas zonas estariam mal polidas. A figura seguinte ilustra o caso em que esse padrão seria corretamente espelhado.



(a) Por polir

(b) Zonas mal polidas

(c) Zona mal polida ao centro

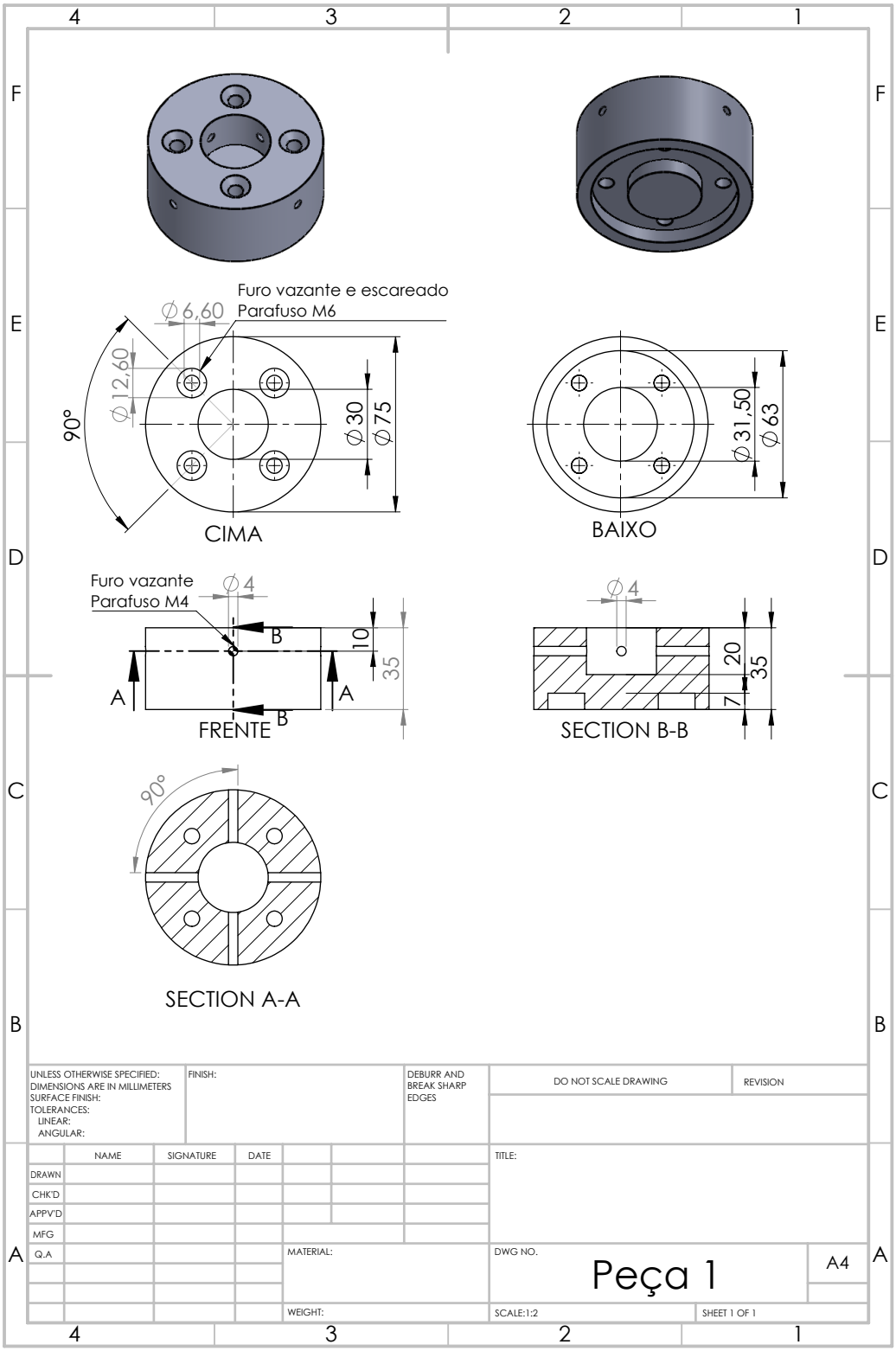
Figura 5.1: Espelhamento de um padrão em peças com vários estados de polimento

Anexos

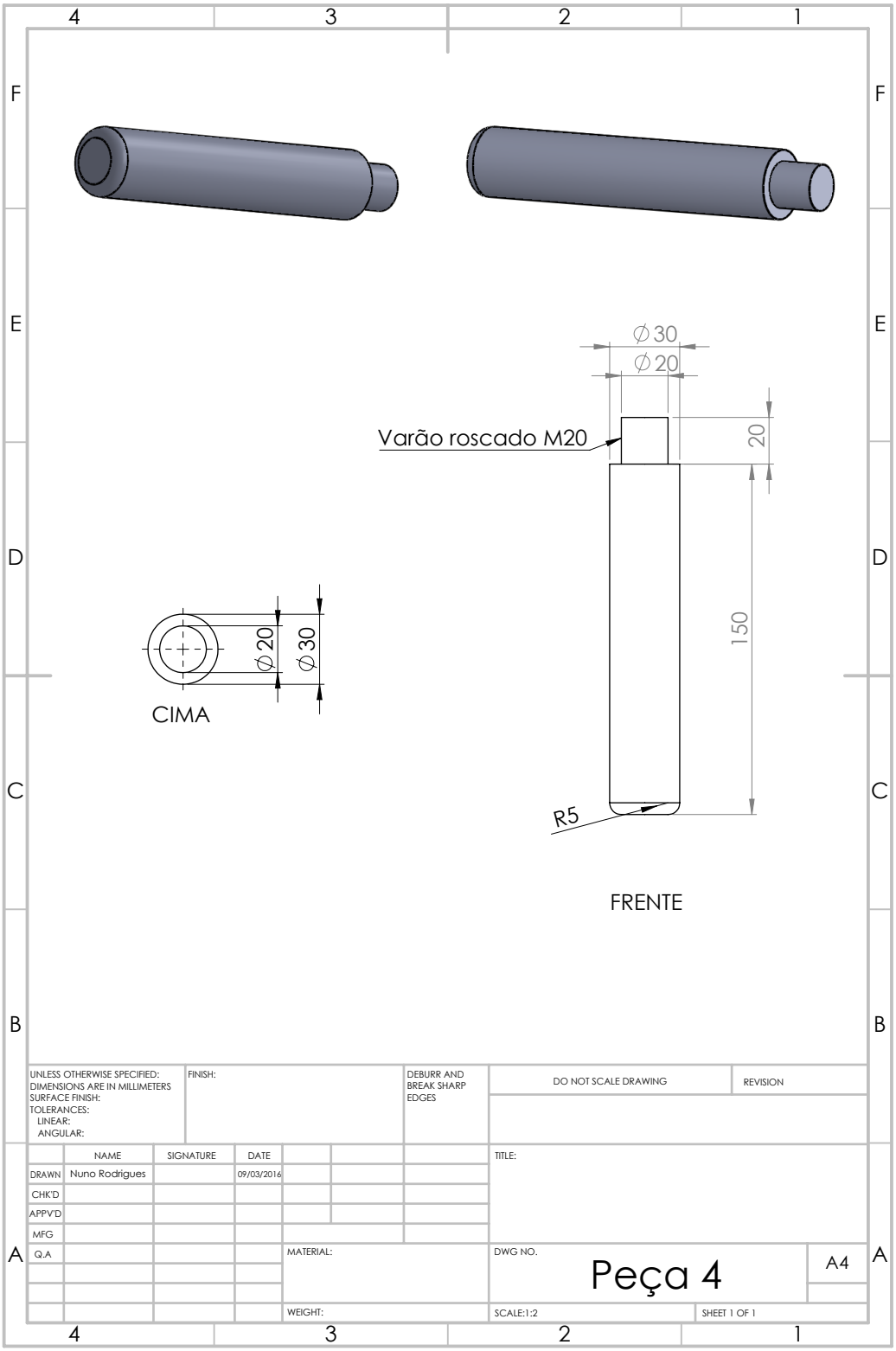
Anexo A

Peças da ferramenta de polimento - Desenho detalhado

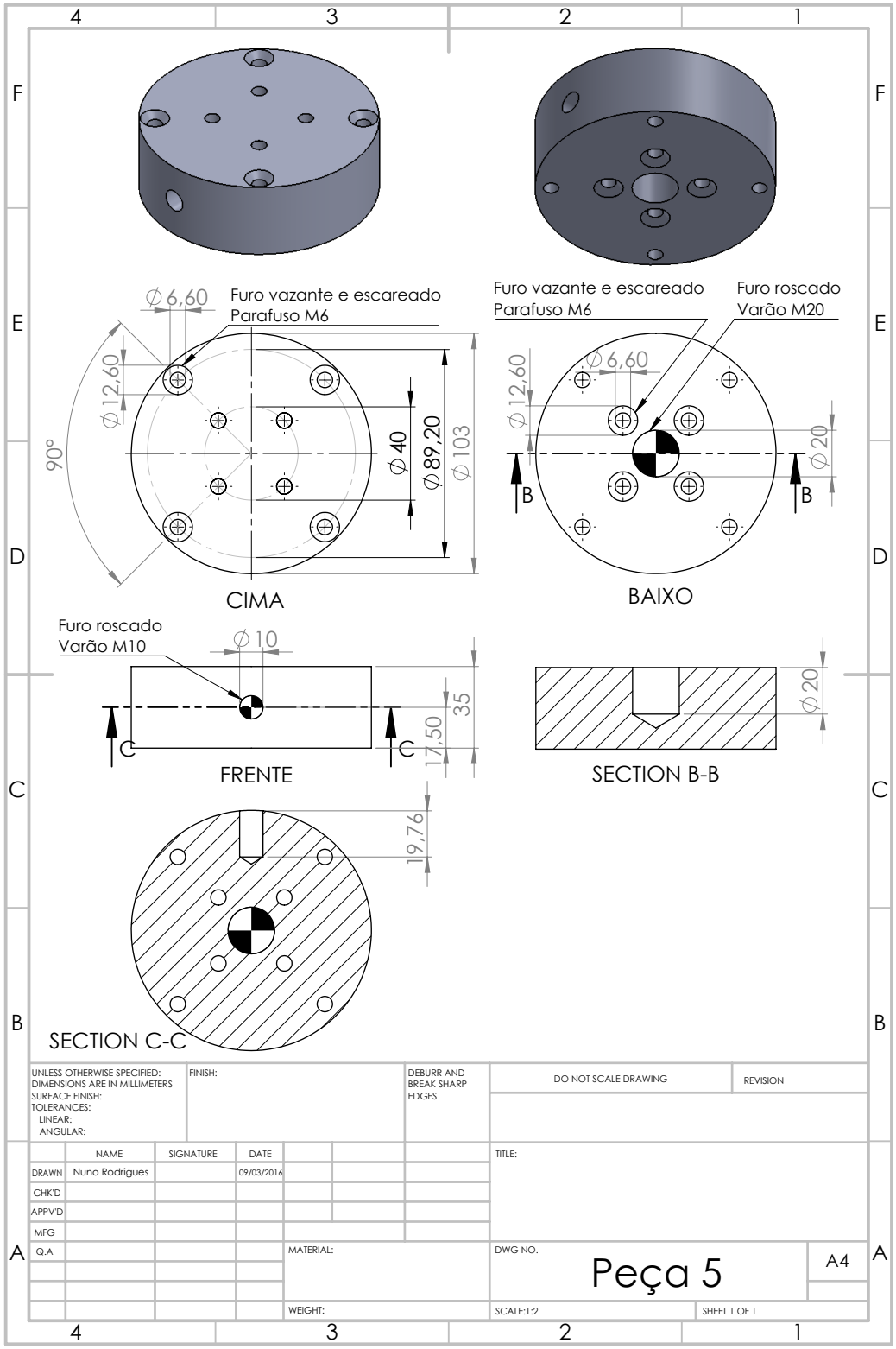
A.1 Peça 1 - Encaixe da pega de polimento manual



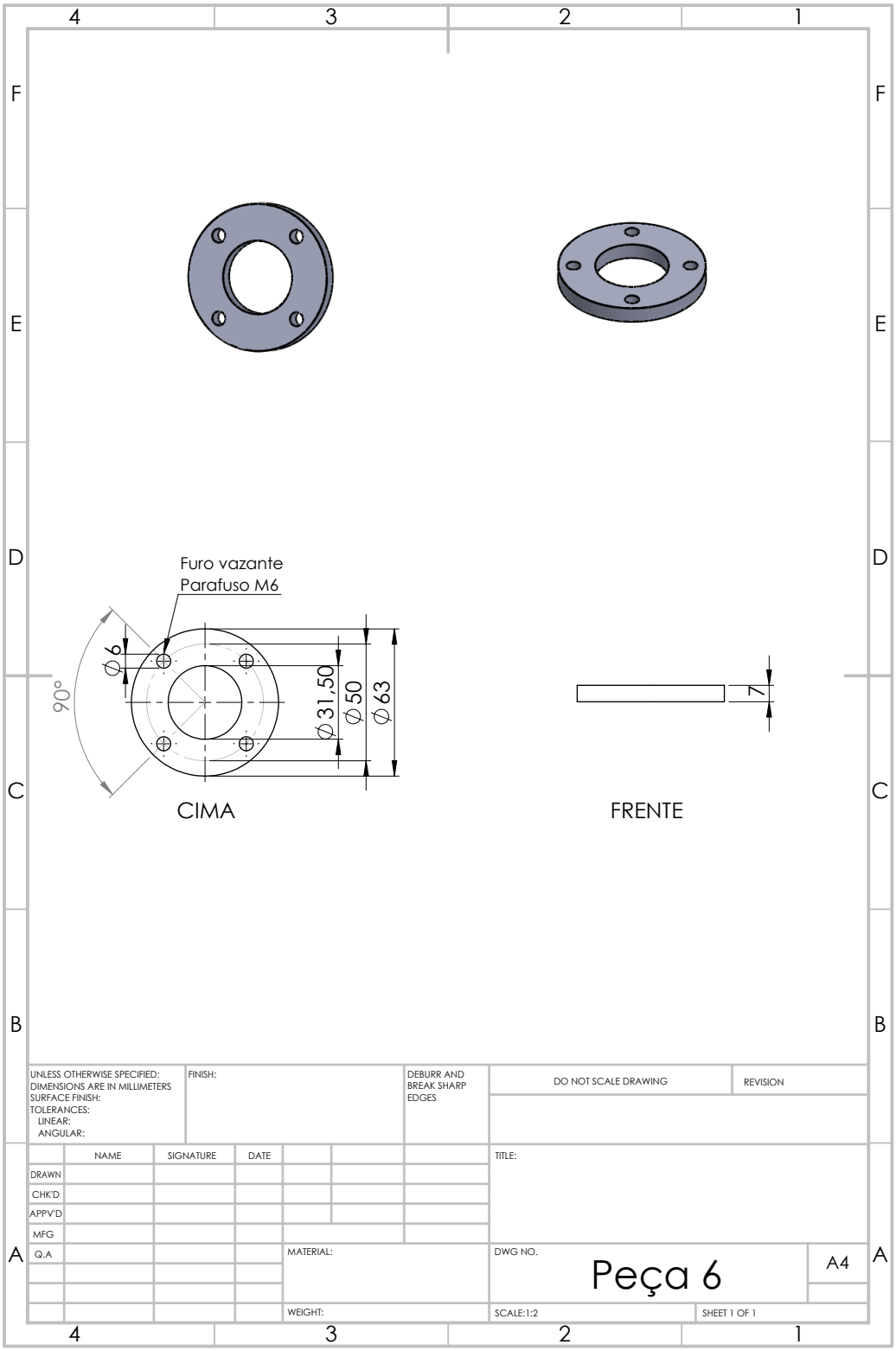
A.2 Peça 4 - Pega da ferramenta de demonstração



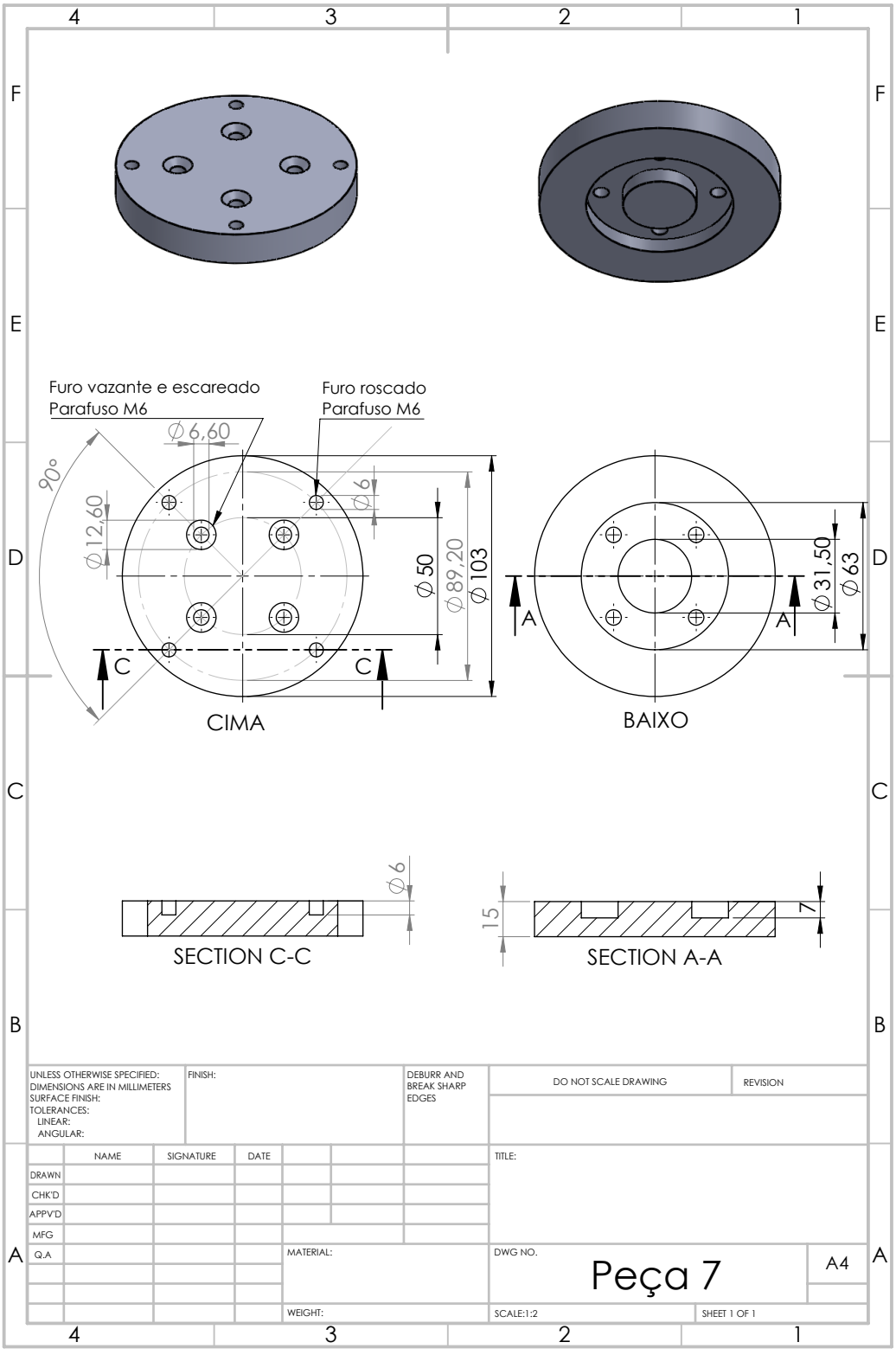
A.3 Peça 5 - Encaixe da pega da ferramenta de demonstração, marcador luminoso e sensor de força



A.4 Peça 6 - Adaptador para encaixar o sensor de força na peça



A.5 Peça 7 - Adaptador para encaixar a peça 5 no manipulador industrial



Anexo B

Código *C++* - Ensinamento por demonstração com controlo de força

B.1 Manipulador industrial *UR5* da *Universal Robots* - Comunicação

B.1.1 Controlo

```
/*sends a message to universal robot*/
string universal_robot::send_to_ur(char *send_buffer, int buffer_len) {

    string msg = this->conn.send_msg(send_buffer, buffer_len);
    if (msg.compare(0, 4, "Erro") == 0) return msg;

    return "ok";
}

/*recieves a message from universal robot*/
string universal_robot::recv_from_ur(char *recv_buffer, int buffer_len) {

    string msg = this->conn.recv_msg(recv_buffer, buffer_len);
    if (msg.compare(0, 4, "Erro") == 0) return msg;

    return "ok";

}

/*
Parameters
q: joint positions - b=true
(q can also be specified as a pose, p) - b=false
a: joint acceleration of leading axis [rad/^s2]
v: joint speed of leading axis [rad/s]
t: time [S]
```

```
r: blend radius [m]
*/
string universal_robot::movej(bool b, double q[6], universal_robot::pose p, double a, double v, double t, double r) {

   ostringstream s;
   char *send_buffer;

   if (b == false) {

      s.str(std::string());

      s << "   movej(p[" << p.X << ", " << p.Y << ", " << p.Z << ", " <<
                     p.Rx << ", " << p.Ry << ", " << p.Rz <<
                "], a=" << a << ", v=" << v << ", r=" << r << ")\n";

      send_buffer = _strdup(s.str().c_str());

      send_to_ur(send_buffer, strlen(send_buffer));

      //printf("program line: %.*s\n", send_buffer, send_buffer + 0);

      Sleep(UR_COMM_DELAY);
   }

   else {

      s.str(std::string());
      s << "   movej([" << q[0] << ", " << q[1] << ", " << q[2] << ", " <<
                    q[3] << ", " << q[4] << ", " << q[5] <<
                "], a=" << a << ", v=" << v << ", t=" << t << ", r=" << r
                    << ")\n";

      send_buffer = _strdup(s.str().c_str());

      send_to_ur(send_buffer, strlen(send_buffer));

      //printf("program line: %.*s\n", send_buffer, send_buffer + 0);

      Sleep(UR_COMM_DELAY);

   }

   return "ok";
}

string universal_robot::movel(universal_robot::pose p) {

   ostringstream s;
```

```
74    char *send_buffer;
75
76    s.precision(7);
77
78    s.str(std::string());
79
80    s << "    movel(p[" << p.X << ", " << p.Y << ", " << p.Z << ", " <<
81                    p.Rx << ", " << p.Ry << ", " << p.Rz <<
82                "], a=" << p.a << ", v=" << p.v << ", r=" << p.r << ")\n";
83
84    send_buffer = _strdup(s.str().c_str());
85
86    send_to_ur(send_buffer, strlen(send_buffer));
87
88    //printf("program line: %.*s\n", send_buffer, send_buffer + 0);
89
90    Sleep(UR_COMM_DELAY);
91
92    return "ok";
93
94 }
95
96 string universal_robot::force_mode(pose task_frame, int selection_vector[6], ↩
       double wrench[6], int type, double limits[6]) {
97
98    ostringstream s;
99    char *send_buffer;
100
101   s.precision(7);
102
103   s.str(std::string());
104
105   s << "    force_mode(p[" << task_frame.X << ", " << task_frame.Y << ", " <<↩
          task_frame.Z << ", " <<
106      task_frame.Rx << ", " << task_frame.Ry << ", " << task_frame.Rz <<
107      "], [" << selection_vector[0] << ", " << selection_vector[1] << ", " <<↩
           selection_vector[2] << ", "
108      << selection_vector[3] << ", " << selection_vector[4] << ", " << ↩
          selection_vector[5] << "]" <<
109      ", [" << wrench[0] << ", " << wrench[1] << ", " << wrench[2] << ", "
110      << wrench[3] << ", " << wrench[4] << ", " << wrench[5] << "]" <<
111      ", " << type <<
112      ", [" << limits[0] << ", " << limits[1] << ", " << limits[2] << ", "
113      << limits[3] << ", " << limits[4] << ", " << limits[5] << "])\n";
114   /*
115   s << "    force_mode(tool_pose(), [" << selection_vector[0] << ", " << ↩
          selection_vector[1] << ", " << selection_vector[2] << ", "
116      << selection_vector[3] << ", " << selection_vector[4] << ", " << ↩
          selection_vector[5] << "]" <<
```

```
117       ", [" << wrench[0] << ", " << wrench[1] << ", " << wrench[2] << ", "
118       << wrench[3] << ", " << wrench[4] << ", " << wrench[5] << "]" <<
119       ", " << type <<
120       ", [" << limits[0] << ", " << limits[1] << ", " << limits[2] << ", "
121       << limits[3] << ", " << limits[4] << ", " << limits[5] << "])\n";
122    */
123    send_buffer = _strdup(s.str().c_str());
124
125    send_to_ur(send_buffer, strlen(send_buffer));
126
127    //printf("program line: %.*s\n", send_buffer, send_buffer + 0);
128
129    Sleep(UR_COMM_DELAY);
130
131    return "ok";
132 }
133
134 /*Resets the robot mode from force mode to normal operation.*/
135 string universal_robot::end_force_mode() {
136
137    ostringstream s;
138    char *send_buffer;
139
140    s.precision(7);
141
142    s.str(std::string());
143
144    s << "   end_force_mode()\n";
145
146    send_buffer = _strdup(s.str().c_str());
147
148    send_to_ur(send_buffer, strlen(send_buffer));
149
150    //printf("program line: %.*s\n", send_buffer, send_buffer + 0);
151
152    Sleep(UR_COMM_DELAY);
153
154    return "ok";
155
156 }
157
158 string universal_robot::end_teach_mode() {
159
160    char *send_buffer = "def end_teach():\n";
161    send_to_ur(send_buffer, strlen(send_buffer));
162    Sleep(UR_COMM_DELAY);
163
164    send_buffer = "    end_teach_mode()\n";
165    send_to_ur(send_buffer, strlen(send_buffer));
```

```
166     Sleep(UR_COMM_DELAY);
167
168     send_buffer = "end\n";
169     send_to_ur(send_buffer, strlen(send_buffer));
170     Sleep(UR_COMM_DELAY);
171
172     return "success";
173
174 }
175
176 string universal_robot::teach_mode() {
177
178     char *send_buffer = "def start_teach():\n";
179     send_to_ur(send_buffer, strlen(send_buffer));
180     Sleep(UR_COMM_DELAY);
181
182     send_buffer = "  def freedriving():\n";
183     send_to_ur(send_buffer, strlen(send_buffer));
184     Sleep(UR_COMM_DELAY);
185
186     send_buffer = "    teach_mode()\n";
187     send_to_ur(send_buffer, strlen(send_buffer));
188     Sleep(UR_COMM_DELAY);
189
190     send_buffer = "    get_actual_tcp_pose()\n";
191     send_to_ur(send_buffer, strlen(send_buffer));
192     Sleep(UR_COMM_DELAY);
193
194     send_buffer = "    sleep(500)\n";
195     send_to_ur(send_buffer, strlen(send_buffer));
196     Sleep(UR_COMM_DELAY);
197
198     send_buffer = "    end_teach_mode()\n";
199     send_to_ur(send_buffer, strlen(send_buffer));
200     Sleep(UR_COMM_DELAY);
201
202     send_buffer = "  end\n";
203     send_to_ur(send_buffer, strlen(send_buffer));
204     Sleep(UR_COMM_DELAY);
205
206     send_buffer = "  while True:\n";
207     send_to_ur(send_buffer, strlen(send_buffer));
208     Sleep(UR_COMM_DELAY);
209
210     send_buffer = "    freedriving()\n";
211     send_to_ur(send_buffer, strlen(send_buffer));
212     Sleep(UR_COMM_DELAY);
213
214     send_buffer = "  end\n";
```

```
215     send_to_ur(send_buffer, strlen(send_buffer));
216     Sleep(UR_COMM_DELAY);
217 
218     send_buffer = "end\n";
219     send_to_ur(send_buffer, strlen(send_buffer));
220     Sleep(UR_COMM_DELAY);
221 
222     return "success";
223 }
224 
225 /*Sends URscrip program header - configuration*/
226 string universal_robot::send_prog_header(double gravity[3], universal_robot::↩
        pose tcp, double payload) {
227 
228     ostringstream s;
229     s.precision(16);
230 
231     char *send_buffer = "def prog():\n";
232     send_to_ur(send_buffer, strlen(send_buffer));
233     //printf("program line: %.*s\n", send_buffer, send_buffer + 0);
234 
235     Sleep(UR_COMM_DELAY);
236 
237     s.str(std::string());
238     s << "    set_gravity([" << gravity[0] << ", " << gravity[1] << ", " << ↩
            gravity[2] << "])\n";
239     send_buffer = _strdup(s.str().c_str());
240     //printf("program line: %.*s\n", send_buffer, send_buffer + 0);
241     //sprintf(send_buffer, "    set_gravity([%f,%f,%f])\n", gravity[0], gravity↩
            [1], gravity[2]);
242     send_to_ur(send_buffer, strlen(send_buffer));
243 
244     Sleep(UR_COMM_DELAY);
245 
246     s.str(std::string());
247     s << "    tcp=p[" << tcp.X << ", " << tcp.Y << ", " << tcp.Z << ", " << tcp↩
            .Rx << ", " << tcp.Ry << ", " << tcp.Rz << "]\n";
248     send_buffer = _strdup(s.str().c_str());
249     //printf("program line: %.*s\n", send_buffer, send_buffer + 0);
250     //sprintf(send_buffer, "    tcp=p[%f,%f,%f,%f,%f,%f]\n", tcp.X, tcp.Y, tcp.↩
            Z, tcp.Rx, tcp.Ry, tcp.Rz);
251     send_to_ur(send_buffer, strlen(send_buffer));
252 
253     Sleep(UR_COMM_DELAY);
254 
255     send_buffer = "    set_tcp(tcp)\n";
256     send_to_ur(send_buffer, strlen(send_buffer));
257     //printf("program line: %.*s\n", send_buffer, send_buffer + 0);
258 
```

```
259     Sleep(UR_COMM_DELAY);
260
261     s.str(std::string());
262     s << "    set_payload(" << payload << ")\n";
263     send_buffer = _strdup(s.str().c_str());
264     //sprintf(send_buffer, "    set_payload(%f)\n", payload);
265     send_to_ur(send_buffer, strlen(send_buffer));
266     //printf("program line: %.*s\n", send_buffer, send_buffer + 0);
267
268     Sleep(UR_COMM_DELAY);
269
270     return "ok";
271 }
272
273 /*Sends URscrip program footer - end*/
274 string universal_robot::send_prog_footer() {
275
276     char *send_buffer = "end\n";
277     send_to_ur(send_buffer, strlen(send_buffer));
278     //printf("program line: %.*s\n", send_buffer, send_buffer + 0);
279
280     Sleep(UR_COMM_DELAY);
281
282     return "ok";
283
284 }
285
286 /*Performs a task from the given points*/
287 string universal_robot::execute_task(vector<universal_robot::pose> points, ↩
        universal_robot::pose tcp, double payload) {
288
289     double gravity[3] = { 9.82, 0, 0 };
290     this->send_prog_header(gravity, tcp, payload);
291
292     double q[6];
293     int i = 0;
294     for(std::vector<pose>::iterator it = points.begin(); it != points.end(); ↩
            ++it) {
295         this->movel(points[i]);
296         cout << endl << ((points.size() - (i + 1)) * UR_COMM_DELAY) / 1000 << "↩
                more seconds to go..." << endl << endl;
297         //this->movej(false, q, points[i], points[i].a, points[i].v, points[i].↩
                t, points[i].r);
298         i++;
299     }
300
301     this->send_prog_footer();
302
303     return "success";
```

```
304
305 }
```

B.1.2 Leitura do estado

```
1  /*Converts to double part of char buffer*/
2  double universal_robot::char_to_double(char *buffer, int pos){
3
4     char aux[sizeof(double)];
5     for (int i = 0; i < sizeof(double); i++) {
6        aux[i] = buffer[pos + (sizeof(double)-1 - i)];
7     }
8
9     return *reinterpret_cast<double *>(&aux[0]);
10 }
11
12 /*Reads actual TCP pose in base referencial*/
13 void universal_robot::get_actual_tcp_pose() {
14
15    char recv_buffer[DEFAULT_BUFFER_LEN];
16
17    int package_len, sub_package_len, search_pointer;
18    UCHAR message_type, package_type;
19    double TCPOffsetX, TCPOffsetY, TCPOffsetZ, TCPOffsetRx, TCPOffsetRy, ↩
          TCPOffsetRz;
20
21    while (1) {
22
23       //printf(" * Package:\n");
24
25       recv_from_ur(recv_buffer, DEFAULT_BUFFER_LEN);
26
27       package_len = ntohl(*(int*)&recv_buffer[0]);
28       //printf(" ** Lenght of overall package: %d\n", package_len);
29
30       message_type = *(UCHAR*)&recv_buffer[4];
31       if (message_type == ROBOT_STATE_PACKAGE) { //Robot state package
32          //printf(" ** Message type: %d - ROBOT_STATE_PACKAGE\n", ↩
                message_type);
33
34          for (search_pointer = 5; search_pointer < package_len; ↩
                search_pointer = search_pointer + sub_package_len) {
35
36             sub_package_len = ntohl(*(int*)&recv_buffer[search_pointer]);
37             //printf(" *** Lenght of sub-package: %d\n", sub_package_len);
38
```

```
39              package_type = *(UCHAR*)&recv_buffer[search_pointer + sizeof(int)];
40              if (package_type == CARTESIAN_INFO) {
41                  //printf(" *** Package type: %d - CARTESIAN_INFO\n", package_type);
42
43                  this->actual_pose.X = this->char_to_double(recv_buffer, search_pointer + sizeof(int) + sizeof(UCHAR));
44                  this->actual_pose.Y = this->char_to_double(recv_buffer, search_pointer + sizeof(int) + sizeof(UCHAR) + sizeof(double));
45                  this->actual_pose.Z = this->char_to_double(recv_buffer, search_pointer + sizeof(int) + sizeof(UCHAR) + 2 * sizeof(double));
46
47                  this->actual_pose.Rx = this->char_to_double(recv_buffer, search_pointer + sizeof(int) + sizeof(UCHAR) + (3 * sizeof(double)));
48                  this->actual_pose.Ry = this->char_to_double(recv_buffer, search_pointer + sizeof(int) + sizeof(UCHAR) + (4 * sizeof(double)));
49                  this->actual_pose.Rz = this->char_to_double(recv_buffer, search_pointer + sizeof(int) + sizeof(UCHAR) + (5 * sizeof(double)));
50
51                  /*
52                  TCPOffsetX = this->char_to_double(recv_buffer, search_pointer + sizeof(int) + sizeof(UCHAR) + (6 * sizeof(double)));
53                  TCPOffsetY = this->char_to_double(recv_buffer, search_pointer + sizeof(int) + sizeof(UCHAR) + (7 * sizeof(double)));
54                  TCPOffsetZ = this->char_to_double(recv_buffer, search_pointer + sizeof(int) + sizeof(UCHAR) + (8 * sizeof(double)));
55
56                  TCPOffsetRx = this->char_to_double(recv_buffer, search_pointer + sizeof(int) + sizeof(UCHAR) + (9 * sizeof(double)));
57                  TCPOffsetRy = this->char_to_double(recv_buffer, search_pointer + sizeof(int) + sizeof(UCHAR) + (10 * sizeof(double)));
58                  TCPOffsetRz = this->char_to_double(recv_buffer, search_pointer + sizeof(int) + sizeof(UCHAR) + (11 * sizeof(double)));
59                  */
60
61                  printf(" *** Actual TCP position\n");
62
63                  printf(" **** X (mm): %f\n", this->actual_pose.X * 1000);
64                  printf(" **** Y (mm): %f\n", this->actual_pose.Y * 1000);
65                  printf(" **** Z (mm): %f\n", this->actual_pose.Z * 1000);
66
67                  printf(" **** Rx (deg): %f\n", rad_to_degree(this->actual_pose.Rx));
```

```
68                printf(" **** Ry (deg): %f\n", rad_to_degree(this->actual_pose↩
                      .Ry));
69                printf(" **** Rz (deg): %f\n", rad_to_degree(this->actual_pose↩
                      .Rz));
70
71
72
73                /*
74                printf(" **** TCPOffsetX: %f\n", TCPOffsetX);
75                printf(" **** TCPOffsetY: %f\n", TCPOffsetY);
76                printf(" **** TCPOffsetZ: %f\n", TCPOffsetZ);
77
78                printf(" **** TCPOffsetRx: %f\n", TCPOffsetRx);
79                printf(" **** TCPOffsetRy: %f\n", TCPOffsetRy);
80                printf(" **** TCPOffsetRz: %f\n", TCPOffsetRz);
81                */
82
83                return;
84             }
85
86          }
87
88       }
89
90    }
91 }
92
93 /*Reads actual joints speeds*/
94 void universal_robot::get_actual_joints_speed() {
95
96    char recv_buffer[DEFAULT_BUFFER_LEN];
97
98    int package_len, sub_package_len, search_pointer;
99    UCHAR message_type, package_type;
100
101   while (1) {
102
103      //printf(" * Package:\n");
104
105      recv_from_ur(recv_buffer, DEFAULT_BUFFER_LEN);
106
107      package_len = ntohl(*(int*)&recv_buffer[0]);
108      //printf(" ** Lenght of overall package: %d\n", package_len);
109
110      message_type = *(UCHAR*)&recv_buffer[4];
111      if (message_type == ROBOT_STATE_PACKAGE) { //Robot state package
112         //printf(" ** Message type: %d - ROBOT_STATE_PACKAGE\n", ↩
                message_type);
113
```

```
114         for (search_pointer = 5; search_pointer < package_len; ↩
                search_pointer = search_pointer + sub_package_len) {
115
116            sub_package_len = ntohl(*(int*)&recv_buffer[search_pointer]);
117            //printf(" *** Lenght of sub-package: %d\n", sub_package_len);
118
119            package_type = *(UCHAR*)&recv_buffer[search_pointer + sizeof(int)↩
                   ];
120            if (package_type == JOINT_DATA) {
121               //printf(" *** Package type: %d - CARTESIAN_INFO\n", ↩
                      package_type);
122
123               printf(" *** Actual joints speeds\n");
124               for (int i = 0; i < 6; i++) {
125                  this->joints_speeds[i] = this->char_to_double(recv_buffer, ↩
                         search_pointer + sizeof(int) + sizeof(UCHAR) + 2*sizeof↩
                         (double) + i * (3 * sizeof(double) + 4 * sizeof(float) ↩
                         + sizeof(UCHAR)));
126                  printf(" **** J[%d]: %f degrees -> %f rads\n", i + 1, this↩
                         ->rad_to_degree(this->joints_speeds[i]), this->↩
                         joints_speeds[i]);
127               }
128
129               return;
130            }
131
132         }
133
134      }
135
136   }
137
138 }
139
140 /*Reads actual joints positions*/
141 void universal_robot::get_actual_joints_pos() {
142
143    char recv_buffer[DEFAULT_BUFFER_LEN];
144
145    int package_len, sub_package_len, search_pointer;
146    UCHAR message_type, package_type;
147
148    while (1) {
149
150       //printf(" * Package:\n");
151
152       recv_from_ur(recv_buffer, DEFAULT_BUFFER_LEN);
153
154       package_len = ntohl(*(int*)&recv_buffer[0]);
```

```
155       //printf(" ** Lenght of overall package: %d\n", package_len);
156
157       message_type = *(UCHAR*)&recv_buffer[4];
158       if (message_type == ROBOT_STATE_PACKAGE) { //Robot state package
159          //printf(" ** Message type: %d - ROBOT_STATE_PACKAGE\n", ↩
              message_type);
160
161          for (search_pointer = 5; search_pointer < package_len; ↩
              search_pointer = search_pointer + sub_package_len) {
162
163             sub_package_len = ntohl(*(int*)&recv_buffer[search_pointer]);
164             //printf(" *** Lenght of sub-package: %d\n", sub_package_len);
165
166             package_type = *(UCHAR*)&recv_buffer[search_pointer + sizeof(int)↩
                 ];
167             if (package_type == JOINT_DATA) {
168                //printf(" *** Package type: %d - CARTESIAN_INFO\n", ↩
                    package_type);
169
170                printf(" *** Actual joints positions\n");
171                for (int i = 0; i < 6; i++) {
172                   this->joints_pos[i] = this->char_to_double(recv_buffer, ↩
                       search_pointer + sizeof(int) + sizeof(UCHAR) + i * (3 *↩
                        sizeof(double) + 4 * sizeof(float) + sizeof(UCHAR)));
173                   printf(" **** J[%d]: %f degrees -> %f rads\n", i+1, this->↩
                       rad_to_degree(this->joints_pos[i]), this->joints_pos[i↩
                       ]);
174                }
175
176                return;
177             }
178
179          }
180
181       }
182
183    }
184
185 }
186
187 /*
188 Waits until the robot reaches the target pose (p, is_pose=true) or joints ↩
     positions (q, is_pose=false)*/
189 void universal_robot::wait_for_next_pose(bool is_pose, pose p, double q[6]) {
190
191    double q_error[6] = { 3.0, 3.0, 3.0, 3.0, 3.0, 3.0 };
192    double pose_error[6] = { 1.0, 1.0, 1.0, 3.0, 3.0, 3.0 };
193
194    double max_joint_error = 0.0001;
```

```
195    double max_pos_error = 0.0002;
196    double max_ang_error = 0.0002;
197
198    int time_joint_adjust = 300;
199
200
201    if (is_pose) {
202
203       while ((std::abs(pose_error[0]) > max_pos_error) || (std::abs(↩
           pose_error[1]) > max_pos_error) ||
204          (std::abs(pose_error[2]) > max_pos_error) || (std::abs(pose_error↩
              [3]) > max_ang_error) ||
205          (std::abs(pose_error[4]) > max_ang_error) || (std::abs(pose_error↩
              [5]) > max_ang_error)) {
206
207          this->get_actual_tcp_pose();
208          this->get_actual_joints_speed();
209
210          pose_error[0] = p.X - this->actual_pose.X;
211          pose_error[1] = p.Y - this->actual_pose.Y;
212          pose_error[2] = p.Z - this->actual_pose.Z;
213          pose_error[3] = p.Rx - this->actual_pose.Rx;
214          pose_error[4] = p.Ry - this->actual_pose.Ry;
215          pose_error[5] = p.Rz - this->actual_pose.Rz;
216
217
218          cout << "pose error: " << endl;
219          //for (int i = 0; i < 6; i++)
220             //cout << pose_error[i] << endl;
221       }
222
223    }
224
225    else {
226
227       q_error[0] = 3.0; q_error[1] = 3.0; q_error[2] = 3.0; q_error[3] = 3.0;↩
           q_error[4] = 3.0; q_error[5] = 3.0;
228       while ((std::abs(q_error[0]) > max_joint_error) || (std::abs(q_error↩
           [1]) > max_joint_error) ||
229          (std::abs(q_error[2]) > max_joint_error) || (std::abs(q_error[3]) > ↩
              max_joint_error) ||
230          (std::abs(q_error[4]) > max_joint_error) || (std::abs(q_error[5]) > ↩
              max_joint_error)) {
231
232          this->get_actual_joints_pos();
233          this->get_actual_joints_speed();
234
235          for (int i = 0; i < 6; i++) {
236             q_error[i] = q[i] - this->joints_pos[i];
```

```
237         }
238
239       }
240
241    }
242
243
244
245    Sleep(time_joint_adjust);
246
247 }
```

B.2 Sensor de força/torque - Comunicação

```
1  string ft_sensor::read_ft() {
2
3     byte request[RESQUEST_ARRAY_LEN];          /* The request data sent to the ↩
          Net F/T. */
4     char resquest_char[RESQUEST_ARRAY_LEN];
5     RESPONSE resp;           /* The structured response received from the Net F↩
          /T. */
6     byte response[RESPONSE_ARRAY_LEN];         /* The raw response data ↩
          received from the Net F/T. */
7     char response_char[RESPONSE_ARRAY_LEN];
8     int i;                   /* Generic loop/array index. */
9     int err;              /* Error status of operations. */
10    char * AXES[] = { "Fx", "Fy", "Fz", "Tx", "Ty", "Tz" };  /* The names of ↩
          the force and torque axes. */
11
12    *(uint16*)&request[0] = htons(0x1234); /* standard header. */
13    *(uint16*)&request[2] = htons(COMMAND_Start); /* per table 9.1 in Net F/T ↩
          user manual. */
14    *(uint32*)&request[4] = htonl(NUM_SAMPLES); /* see section 9.1 in Net F/T ↩
          user manual. */
15
16    for (unsigned int i = 0; i < RESQUEST_ARRAY_LEN; i++)
17       resquest_char[i] = (char)request[i]; // Byte to char conversion
18
19    string msg = this->conn.send_msg(resquest_char, RESQUEST_ARRAY_LEN);
20    if (msg.compare(0, 4, "Erro") == 0) return msg;
21
22    msg = this->conn.recv_msg(response_char, RESPONSE_ARRAY_LEN);
23    if (msg.compare(0, 4, "Erro") == 0) {
24       return msg;
25    }
26
```

```
27    for (unsigned int i = 0; i < RESPONSE_ARRAY_LEN; i++)
28       response[i] = (byte)response_char[i]; // Char to byte conversion
29
30    resp.rdt_sequence = ntohl(*(uint32*)&response[0]);
31    resp.ft_sequence = ntohl(*(uint32*)&response[4]);
32    resp.status = ntohl(*(uint32*)&response[8]);
33
34    for (i = 0; i < 6; i++) {
35       resp.FTData[i] = ntohl(*(int32*)&response[12 + i * 4]);
36    }
37
38    /* Output the response data. */
39    printf("Status: 0x%08x\n", resp.status);
40    for (i = 0; i < 6; i++)
41    {
42       this->ft_values[i] = resp.FTData[i] / COUNTS_PER_FORCE;
43       printf("%s: %.4f %s\n", AXES[i], this->ft_values[i], FORCE_UNIT);
44    }
45
46    return msg;
47 }
48 string ft_sensor::remove_bias() {
49
50    byte request[RESQUEST_ARRAY_LEN];/* The request data sent to the Net F/T. ↩
          */
51    char resquest_char[RESQUEST_ARRAY_LEN];
52
53    *(uint16*)&request[0] = htons(0x1234); /* standard header. */
54    *(uint16*)&request[2] = htons(COMMAND_Bias); /* per table 9.1 in Net F/T ↩
          user manual. */
55
56    for (unsigned int i = 0; i < RESQUEST_ARRAY_LEN; i++)
57       resquest_char[i] = (char)request[i]; //Byte to char conversion
58
59    string msg= this->conn.send_msg(resquest_char, RESQUEST_ARRAY_LEN);
60
61    return msg;
62 }
```

B.3 *6DMimic* - Comunicação

```
1  void sixDmimic::start_and_sync(const char *file_name) {
2
3     char buf[BUFLEN];
4     int npos;
5     double a, x, y, b, delay;
```

```
6   double first_frame, offset;
7
8   string s_buf = "START6DM ";
9   s_buf.append(string(file_name));
10  cout << "Sent to 6D-Mimic: " << s_buf << endl;
11
12  npos = s_buf.length();
13  strncpy(buf, s_buf.c_str(), sizeof(buf));
14  int sizedest = sizeof(dest);
15
16  int ret = sendto(s, buf, npos, 0, (sockaddr*)&dest, sizedest);
17  if (ret == -1) {
18     printf("\nSend Error Code : \n");
19  }
20
21
22  fflush(stdout);
23
24  //limpar o buffer com o memset - Acrescentar o ócdigo
25
26  //try to receive some data, this is a blocking call
27  if ((recvfrom(s, buf, BUFLEN, 0, (sockaddr*)&dest, &slen)) == SOCKET_ERROR↩
        )
28  {
29     printf("recvfrom() failed with error code : %d", WSAGetLastError());
30     exit(EXIT_FAILURE);
31  }
32
33  //t1
34  a = std::chrono::duration_cast<std::chrono::microseconds>(std::chrono::↩
        system_clock::now().time_since_epoch()).count();
35
36  s_buf = string(buf);
37
38  //t0
39  x = stod(s_buf.substr(10));
40
41  offset = a - x;
42
43  fflush(stdout);
44  //limpar o buffer com o memset - Acrescentar o ócdigo
45  //try to receive some data, this is a blocking call
46  if ((recv_len = recvfrom(s, buf, BUFLEN, 0, (sockaddr*)&dest, &slen)) == ↩
        SOCKET_ERROR)
47  {
48     printf("recvfrom() failed with error code : %d", WSAGetLastError());
49     exit(EXIT_FAILURE);
50  }
51
```

```
52    s_buf = string(buf);
53
54    first_frame = stod(s_buf.substr(15));
55
56    cout << " * Start and sync info\n\n";
57    cout << " ** Start" << endl;
58    printf("6Dmimic time x: %f\n", x);
59    printf("Local time a:   %f\n", a);
60    printf("6DMimic clock offset (ms): %f\n", offset / 1000);
61    cout << " ** 1sFrame" << endl;
62    printf("6Dmimic time first frame: %f\n", first_frame);
63
64    fstream fs;
65    fs.open("6DM_start_sync_info.txt", ios::out | ios::trunc);
66    if (!fs)
67    {
68       std::cerr << "Cannot open the output file: 6DM_start_sync_info" << std↩
          ::endl;
69       return;
70    }
71    fs << "clock_offset: " << offset << endl;
72    fs << "first_frame: " << first_frame << endl;
73    fs.close();
74
75    //Sync with NTP
76
77    /*
78    char buf[BUFLEN];
79    int npos;
80    double a, x, y, b, delay;
81
82    string s_buf = "START6DM ";
83    s_buf.append(string(file_name));
84    cout << "Sent to 6D-Mimic: " << s_buf << endl;
85
86    npos = s_buf.length();
87    strncpy(buf, s_buf.c_str(), sizeof(buf));
88    int sizedest = sizeof(dest);
89
90    a = std::chrono::duration_cast<std::chrono::microseconds>(std::chrono::↩
       system_clock::now().time_since_epoch()).count();
91
92    int ret = sendto(s, buf, npos, 0, (sockaddr*)&dest, sizedest);
93    if (ret == -1) {
94       printf("\nSend Error Code : \n");
95    }
96
97    fflush(stdout);
98    //áNecessrio limpar o buffer com o memset - Acrecentar o ócdigo
```

```
99
100   //try to receive some data, this is a blocking call
101   if ((recvfrom(s, buf, BUFLEN, 0, (sockaddr*)&dest, &slen)) == SOCKET_ERROR↩
          )
102   {
103       printf("recvfrom() failed with error code : %d", WSAGetLastError());
104       exit(EXIT_FAILURE);
105   }
106
107   s_buf = string(buf);
108
109   x = stod(s_buf.substr(10));
110
111   fflush(stdout);
112   //clear the buffer by filling null, it might have previously received data
113   //limpar o buffer com o memset - Acrescentar o ócdigo
114
115   //try to receive some data, this is a blocking call
116   if ((recv_len = recvfrom(s, buf, BUFLEN, 0, (sockaddr*)&dest, &slen)) == ↩
          SOCKET_ERROR)
117   {
118       printf("recvfrom() failed with error code : %d", WSAGetLastError());
119       exit(EXIT_FAILURE);
120   }
121
122   b = std::chrono::duration_cast<std::chrono::microseconds>(std::chrono::↩
          system_clock::now().time_since_epoch()).count();
123
124   s_buf = string(buf);
125
126   y = stod(s_buf.substr(15));
127
128   cout << " * Start and sync info\n\n";
129   cout << " ** Start" << endl;
130   printf("Local time a:   %f\n", a);
131   printf("6Dmimic time x: %f\n", x);
132   cout << " ** 1sFrame" << endl;
133   printf("6Dmimic time y: %f\n", y);
134   printf("Local time b:   %f\n", b);
135
136   delay = (b - a) - (y - x);
137   offset = x - (a + (delay / 2));
138
139   printf("b-a (ms): %f\n", (b - a) / 1000);
140   printf("y-x (ms): %f\n", (y - x)/1000);
141
142   cout << "Communication delay (ms): " << delay/1000 << endl;
143   printf("6DMimic clock offset (ms): %f\n", offset/1000);
144
```

```
145     std::fstream fs;
146     fs.open("sync_log.txt", std::fstream::in | std::fstream::out | std::↩
            fstream::app);
147     fs << delay / 1000 << "                          " << offset / 1000 << "\n";
148     fs.close();
149 
150     return;
151     */
152 }
153 
154 
155 int sixDmimic::stop() {
156 
157     char buf[BUFLEN];
158     string s_buf;
159     int npos;
160 
161     printf("Sent to 6D-Mimic: STOP6DM\n");
162 
163     int ret = sendto(s, "STOP6DM", 7, 0, (sockaddr*)&dest, sizeof(dest));
164     if (ret == -1) {
165         printf("\nSend Error Code : \n");
166     }
167 
168     cout << "Recieved from 6DM: ";
169 
170     fflush(stdout);
171 
172     //try to receive some data, this is a blocking call
173     if ((recv_len = recvfrom(s, buf, BUFLEN, 0, (sockaddr*)&dest, &slen)) == ↩
            SOCKET_ERROR)
174     {
175         printf("recvfrom() failed with error code : %d", WSAGetLastError());
176         exit(EXIT_FAILURE);
177     }
178 
179     s_buf = string(buf);
180 
181     if (s_buf.compare("6DM_END") == 0) {
182         cout << "END" << endl;
183         return 0;
184     }
185 
186     else if (s_buf.compare("6DM_END_NOK") == 0) {
187         cout << "END NOK" << endl;
188         return -1;
189     }
190 
191     cout << "Trash" << endl;
```

```
192     return -2;
193
194 }
```

B.4 Função desenvolvida para transformar os pontos da trajetória da zona de demonstração para a zona de atuação e suavizá-la antes de ser enviada para o *UR5*

```
1  void demo::transform_6DMimic_points(double a, double r, double posToleration,↩
       double angleToleration, double vel_adjust, universal_robot::pose ↩
      p_calib_6DM, universal_robot::pose p_calib_MOTOMAN, universal_robot::pose↩
       p_calib_UR) {
2
3
4     //Calculo da transformacao homogenea da esfera para o TCP - H_1_2
5     Matrix4d H_0_1_calib, H_0_2_calib, H_1_2;
6     Matrix3d R_0_1_calib, R_0_2_calib;
7     AngleAxisd aa;
8
9     R_0_1_calib = AngleAxisd(p_calib_6DM.Rz, Vector3d::UnitZ())
10       * AngleAxisd(p_calib_6DM.Ry, Vector3d::UnitY())
11       * AngleAxisd(p_calib_6DM.Rx, Vector3d::UnitX());
12    aa.fromRotationMatrix(R_0_1_calib);
13
14    H_0_1_calib = demo::get_HomeogeneousTransform(aa,
15                                                  p_calib_6DM.X,
16                                                  p_calib_6DM.Y,
17                                                  p_calib_6DM.Z);
18
19    R_0_2_calib = AngleAxisd(p_calib_MOTOMAN.Rz, Vector3d::UnitZ())
20       * AngleAxisd(p_calib_MOTOMAN.Ry, Vector3d::UnitY())
21       * AngleAxisd(p_calib_MOTOMAN.Rx, Vector3d::UnitX());
22    aa.fromRotationMatrix(R_0_2_calib);
23
24    H_0_2_calib = demo::get_HomeogeneousTransform(aa,
25                                                  p_calib_MOTOMAN.X,
26                                                  p_calib_MOTOMAN.Y,
27                                                  p_calib_MOTOMAN.Z);
28
29    H_1_2 = H_0_1_calib.inverse()*H_0_2_calib;
30
31
32    //Calculo da çãtransformao homogenea do TCP do MOTOMAN para o TCP do UR - ↩
          H_2_4
33    Matrix4d H_3_0, H_3_4_calib, H_2_4;
```

```
34    Matrix3d R_3_0, R_3_4_calib, R_2_4;
35
36    R_3_0 = AngleAxisd(universal_robot::degree_to_rad(0), Vector3d::UnitZ())
37       * AngleAxisd(universal_robot::degree_to_rad(90), Vector3d::UnitY())
38       * AngleAxisd(universal_robot::degree_to_rad(0), Vector3d::UnitX());
39    aa.fromRotationMatrix(R_3_0);
40    H_3_0 = demo::get_HomeogeneousTransform(aa,
41                                            0,
42                                            0,
43                                            0);
44
45    /*Vector3d vr_3_4 = Vector3d(p_calib_UR.Rx, p_calib_UR.Ry, p_calib_UR.Rz);
46    double norm = vr_3_4.norm();
47    Vector3d vr_3_4_normalized = vr_3_4.normalized();
48    AngleAxisd aa_3_4(norm, vr_3_4_normalized);
49
50    H_3_4_calib = demo::get_HomeogeneousTransform(aa_3_4,
51                                                  p_calib_UR.X,
52                                                  p_calib_UR.Y,
53                                                  p_calib_UR.Z);
54
55    H_2_4 = H_0_2_calib.inverse()*H_3_0.inverse()*H_3_4_calib;*/
56
57    Vector3d v_motoman_calib(p_calib_MOTOMAN.X, p_calib_MOTOMAN.Y, ↩
          p_calib_MOTOMAN.Z);
58    v_motoman_calib = R_3_0 * v_motoman_calib;
59
60    Vector3d translacao = v_motoman_calib - Vector3d(p_calib_UR.X, p_calib_UR.↩
          Y, p_calib_UR.Z);
61
62    //Transformacao de todos os pontos
63    Matrix4d H_0_1, H_3_4;
64    Matrix3d R_0_1, R_0_1_ant, R_3_4;
65    Quaterniond q1, q2, q3;
66    AngleAxisd ang_var;
67    Vector3d v, deslocamento;
68    vector<universal_robot::pose> filtred;
69    double norm, x_sum, y_sum, z_sum, Rx_sum, Ry_sum, Rz_sum, v_sum, ang_sum;
70    int acumul_points = 0;
71    x_sum = 0; y_sum = 0; z_sum = 0; Rx_sum = 0; Ry_sum = 0; v_sum = 0; ↩
          ang_sum = 0;
72    Rz_sum = 0;
73    AngleAxisd ang_avg;
74
75    //ciclo de ajuste da trajetoria
76    for (int c = 0; c < p_segs_file.size(); c++) {
77
78       R_0_1 = AngleAxisd(p_segs_file[c].Rz, Vector3d::UnitZ())
79          * AngleAxisd(p_segs_file[c].Ry, Vector3d::UnitY())
```

```
80          * AngleAxisd(p_segs_file[c].Rx, Vector3d::UnitX());
81      aa.fromRotationMatrix(R_0_1);
82
83      if (c == 0) {
84          filtred.push_back(p_segs_file[c]);
85      }
86
87      else if (c > 0 && c < p_segs_file.size() - 1) {
88
89          R_0_1_ant = AngleAxisd(filtred[filtred.size() - 1].Rz, Vector3d::↩
                UnitZ())
90              * AngleAxisd(filtred[filtred.size() - 1].Ry, Vector3d::UnitY())
91              * AngleAxisd(filtred[filtred.size() - 1].Rx, Vector3d::UnitX());
92
93          q1 = Quaterniond(R_0_1_ant);
94          q2 = Quaterniond(R_0_1);
95          q3 = q1*q2.inverse();
96          ang_var = AngleAxisd(q3);
97
98          deslocamento = Vector3d(p_segs_file[c].X, p_segs_file[c].Y, ↩
                p_segs_file[c].Z) -
99                      Vector3d(filtred[filtred.size() - 1].X, filtred[filtred.↩
                            size() - 1].Y, filtred[filtred.size() - 1].Z);
100
101         norm = sqrt(deslocamento.x()*deslocamento.x() + deslocamento.y()*↩
                deslocamento.y() + deslocamento.z()*deslocamento.z());
102
103         acumul_points++;
104
105         //Angle axis tem uma unica çãsoluo para o eixo (excepto para o sinal↩
                ), sendo o sinal do angulo determinado pelo sinal do eixo
106         if (universal_robot::rad_to_degree(std::abs(ang_var.angle())) >= ↩
                angleToleration || norm >= posToleration) {
107
108             //print_pose(p_segs_file[c]);
109
110             x_sum += p_segs_file[c].X; y_sum += p_segs_file[c].Y; z_sum += ↩
                    p_segs_file[c].Z;
111             v_sum += p_segs_file[c].v;
112
113
114             if (aa.angle() < 0) {
115                 print_AngleAxis(aa);
116                 aa.axis() = -1 * aa.axis();
117                 aa.angle() = -1 * aa.angle();
118                 print_AngleAxis(aa);
119                 system("pause");
120             }
121
```

```
122             Rx_sum += aa.axis().x(); Ry_sum += aa.axis().y(); Rz_sum += aa.↩
                    axis().z();
123             ang_sum += aa.angle();
124 
125 
126             ang_avg = AngleAxisd(ang_sum / acumul_points, Vector3d(Rx_sum / ↩
                    acumul_points, Ry_sum / acumul_points, Rz_sum / acumul_points↩
                    ).normalized());
127             Matrix3d ang_avg_mat = ang_avg.toRotationMatrix();
128 
129             p_segs_file[c].X = x_sum / acumul_points;
130             p_segs_file[c].Y = y_sum / acumul_points;
131             p_segs_file[c].Z = z_sum / acumul_points;
132             p_segs_file[c].Rx = ang_avg_mat.eulerAngles(2, 1, 0).z();
133             p_segs_file[c].Ry = ang_avg_mat.eulerAngles(2, 1, 0).y();
134             p_segs_file[c].Rz = ang_avg_mat.eulerAngles(2, 1, 0).x();
135             p_segs_file[c].v = v_sum / acumul_points;
136 
137             //cout << "Resultante da media: " << endl;
138             //print_pose(p_segs_file[c]);
139             //print_AngleAxis(AngleAxisd(averageRot));
140             //system("pause");
141 
142             filtred.push_back(p_segs_file[c]);
143 
144             acumul_points = 0; x_sum = 0; y_sum = 0; z_sum = 0; v_sum = 0;
145             Rx_sum = 0; Ry_sum = 0; Rz_sum = 0; ang_sum = 0;
146         }
147         else {
148 
149             x_sum += p_segs_file[c].X; y_sum += p_segs_file[c].Y; z_sum += ↩
                    p_segs_file[c].Z;
150             v_sum += p_segs_file[c].v;
151 
152             if (aa.angle() < 0) {
153                 print_AngleAxis(aa);
154                 aa.axis() = -1 * aa.axis();
155                 aa.angle() = -1 * aa.angle();
156                 print_AngleAxis(aa);
157                 system("pause");
158             }
159 
160             Rx_sum += aa.axis().x(); Ry_sum += aa.axis().y(); Rz_sum += aa.↩
                    axis().z();
161             ang_sum += aa.angle();
162 
163             //print_AngleAxis(AngleAxisd(R_0_1));
164             //print_pose(p_segs_file[c]);
165         }
```

```
166
167         }
168
169         else if (c == p_segs_file.size() - 1) {
170
171             /*acumul_points++;
172
173             x_sum += p_segs_file[c].X; y_sum += p_segs_file[c].Y; z_sum += ↩
                    p_segs_file[c].Z;
174             v_sum += p_segs_file[c].v;
175
176             p_segs_file[c].X = x_sum / acumul_points;
177             p_segs_file[c].Y = y_sum / acumul_points;
178             p_segs_file[c].Z = z_sum / acumul_points;*/
179             /*p_segs_file[c].Rx = avg_rot_v.z();
180             p_segs_file[c].Ry = avg_rot_v.y();
181             p_segs_file[c].Rz = avg_rot_v.x();*/
182             //p_segs_file[c].v = v_sum / acumul_points;
183
184             filtred.push_back(p_segs_file[c]);
185
186             acumul_points = 0; x_sum = 0; y_sum = 0; z_sum = 0; v_sum = 0;
187             Rx_sum = 0; Ry_sum = 0; Rz_sum = 0; ang_sum = 0;
188
189         }
190
191     }
192
193     this->p_segs_file = filtred;
194
195     //ciclo de ajuste da velocidade
196     for (int c = 0; c < p_segs_file.size(); c++) {
197         p_segs_file[c].v = p_segs_file[c].v * vel_adjust;
198     }
199
200     for (int i = 0; i < p_segs_file.size(); i++) {
201
202         R_0_1 = AngleAxisd(p_segs_file[i].Rz, Vector3d::UnitZ())
203             * AngleAxisd(p_segs_file[i].Ry, Vector3d::UnitY())
204             * AngleAxisd(p_segs_file[i].Rx, Vector3d::UnitX());
205         aa.fromRotationMatrix(R_0_1);
206
207
208         H_0_1 = demo::get_HomeogeneousTransform(aa,
209                                         p_segs_file[i].X,
210                                         p_segs_file[i].Y,
211                                         p_segs_file[i].Z);
212
213
```

```
214         //H_3_4 = H_3_0*H_0_1*H_1_2*H_2_4;
215         H_3_4 = H_3_0*H_0_1*H_1_2;
216 
217         R_3_4(0, 0) = H_3_4(0, 0);
218         R_3_4(0, 1) = H_3_4(0, 1);
219         R_3_4(0, 2) = H_3_4(0, 2);
220         R_3_4(1, 0) = H_3_4(1, 0);
221         R_3_4(1, 1) = H_3_4(1, 1);
222         R_3_4(1, 2) = H_3_4(1, 2);
223         R_3_4(2, 0) = H_3_4(2, 0);
224         R_3_4(2, 1) = H_3_4(2, 1);
225         R_3_4(2, 2) = H_3_4(2, 2);
226 
227         //v = R_3_4.eulerAngles(2, 1, 0);
228         aa.fromRotationMatrix(R_3_4);
229         v = aa.angle()*aa.axis();
230 
231         this->p_segs_file[i].X = H_3_4(0, 3) - translacao.x();
232         this->p_segs_file[i].Y = H_3_4(1, 3) - translacao.y();
233         this->p_segs_file[i].Z = H_3_4(2, 3) - translacao.z();
234 
235         this->p_segs_file[i].Rx = v.x();
236         this->p_segs_file[i].Ry = v.y();
237         this->p_segs_file[i].Rz = v.z();
238 
239         this->p_segs_file[i].a = a;
240         this->p_segs_file[i].r = r;
241         this->p_segs_file[i].t = 0;
242 
243         print_pose(p_segs_file[i]);
244 
245     }
246 
247 
248 }
```

Anexo C

Código *C++* - Inspeção por visão artificial

Neste anexo apresenta-se o código principal desenvolvido relativo à inspeção por visão artificial de peças altamente refletoras e espelhadas.

C.1 Função desenvolvida para segmentar os defeitos, usando o operador Laplaciano

```
void defects_segm_laplacian() {

    Mat image_gb, bin_img, edges, edges_gb, abs_edges;
    int kernel_size = g_kernel_size;
    int scale = 1;
    int delta = 0;
    int ddepth = CV_16S;
    Mat element = getStructuringElement(MORPH_RECT, Size(1 * 2 + 1, 1 * 2 + 1)↩
        , Point(1, 1));

    /// Remove noise by blurring with a Gaussian filter
    GaussianBlur(image, image_gb, gauss_blur_kernel_size, 0, 0, BORDER_DEFAULT↩
        );
    // Create a window for display.
    if (show_images) namedWindow("After applying Gaussian filter", ↩
        WINDOW_AUTOSIZE);
    // Show our image inside it.
    if (show_images) imshow("After applying Gaussian filter", image_gb);

    /*Laplacian ---------------------------------------------------*/
    Laplacian(image_gb, edges, ddepth, kernel_size, scale, delta, ↩
        BORDER_DEFAULT);
    convertScaleAbs(edges, abs_edges);
```

```
20    if (show_images) namedWindow("After applying Laplacian", WINDOW_AUTOSIZE);
21    if (show_images) imshow("After applying Laplacian", abs_edges);
22
23    int th = 20;
24    threshold(abs_edges, bin_img, th, 255, THRESH_BINARY);
25    if (show_images) namedWindow("Laplacian - Image binarization - Threshold:
          " + to_string(th), WINDOW_AUTOSIZE);
26    if (show_images) imshow("Laplacian - Image binarization - Threshold: " +
          to_string(th), bin_img);
27
28    /*MORPH_CLOSE*/
29    morphologyEx(bin_img, defects, MORPH_CLOSE, element);
30    if (show_images) namedWindow("Laplacian - After closing", WINDOW_AUTOSIZE)
          ;
31    if (show_images) imshow("Laplacian - After closing", defects);
32
33    /*Crops image by the mask*/
34    bitwise_and(mask, defects, croppedImage);
35    if (show_images) namedWindow("Cropped image using the mask",
          WINDOW_AUTOSIZE);
36    if (show_images) imshow("Cropped image using the mask", croppedImage);
37
38    defects = croppedImage;
39
40 }
```

C.2 Função desenvolvida para segmentar os defeitos, usando o gradiente morfológico

```
1  void defects_segm_gradient() {
2
3     Mat edges, abs_edges, image_gb, bin_img;
4
5     Mat element = getStructuringElement(MORPH_RECT, Size(1 * 2 + 1, 1 * 2 + 1)
          , Point(1, 1));
6
7     /// Remove noise by blurring with a Gaussian filter
8     GaussianBlur(image, image_gb, Size(5, 5), 0, 0, BORDER_DEFAULT);
9     if (show_images) namedWindow("After applying Gaussian filter",
          WINDOW_AUTOSIZE);
10    if (show_images) imshow("After applying Gaussian filter", image_gb);
11
12    morphologyEx(image_gb, edges, MORPH_GRADIENT, element);
13    convertScaleAbs(edges, abs_edges);
```

```
14    if (show_images) namedWindow("After applying Morphological gradient", ↩
          WINDOW_AUTOSIZE);
15    if (show_images) imshow("After applying Morphological gradient", abs_edges↩
          );
16
17    int th = 20;
18    threshold(abs_edges, bin_img, th, 255, THRESH_BINARY);
19    if (show_images) namedWindow("Morphological gradient - Image binarization ↩
          - Threshold: " + to_string(th), WINDOW_AUTOSIZE);
20    if (show_images) imshow("Morphological gradient - Image binarization - ↩
          Threshold: " + to_string(th), bin_img);
21
22    /*MORPH_CLOSE*/
23    morphologyEx(bin_img, defects, MORPH_CLOSE, element);
24    if (show_images) namedWindow("Morphological gradient - After closing", ↩
          WINDOW_AUTOSIZE);
25    if (show_images) imshow("Morphological gradient - After closing", defects)↩
          ;
26
27    /*Crops image by the mask*/
28    bitwise_and(mask, defects, croppedImage);
29    if (show_images) namedWindow("Cropped image using the mask", ↩
          WINDOW_AUTOSIZE);
30    if (show_images) imshow("Cropped image using the mask", croppedImage);
31
32    defects = croppedImage;
33 }
```

C.3 Função desenvolvida para identificar os defeitos

```
1  int defects_detection(string win_name, int &max_area_i) {
2
3     vector<vector<Point> > contours;
4     vector<Vec4i> hierarchy;
5     Scalar rect_color;
6     Mat image_rgb;
7     int total_defs = 0;
8     max_area_i = 0;
9
10    /// Find contours
11    findContours(defects, contours, hierarchy, CV_RETR_TREE, ↩
          CV_CHAIN_APPROX_SIMPLE, Point(0, 0));
12    vector<Moments> mu(contours.size());
13    vector<bool> defects_rejection(contours.size());
14
15    /// Get the moments
```

```
16    for (int i = 0; i < contours.size(); i++)
17    {
18       mu[i] = moments(contours[i], false);
19       // Reject defects
20       if (mu[i].m00 < MIN_AREA_DEF)
21          defects_rejection[i] = true;
22       else {
23          defects_rejection[i] = false;
24          total_defs++;
25       }
26
27       if (mu[i].m00 > max_area_i)
28          max_area_i = mu[i].m00;
29    }
30
31    vector<Point2f> mc(contours.size());
32    vector<RotatedRect> minRect(contours.size());
33    for (int i = 0; i < contours.size(); i++)
34    {
35       /// Find the rotated rectangles for each contour
36       minRect[i] = minAreaRect(Mat(contours[i]));
37       ///  Get the mass centers:
38       mc[i] = Point2f(mu[i].m10 / mu[i].m00, mu[i].m01 / mu[i].m00);
39
40    }
41
42    /// Draw contours + rotated rects + Mass centers points
43    Mat drawing = Mat::zeros(defects.size(), CV_8UC3);
44    cvtColor(image, image_rgb, COLOR_GRAY2BGR);
45    for (int i = 0; i< contours.size(); i++)
46    {
47
48       // contour
49       drawContours(drawing, contours, i, Scalar(255, 255, 255), 1, 8, vector<↩
             Vec4i>(), 0, Point());
50       // mass center
51       //circle(drawing, mc[i], 1, Scalar(0, 255, 0), 2, 8, 0);
52       // rotated rectangle
53       Point2f rect_points[4]; minRect[i].points(rect_points);
54
55       if (defects_rejection[i] == false)
56          rect_color = Scalar(0, 0, 255); // True defects are red
57       else
58          rect_color = Scalar(0, 223, 15); // True defects are green
59
60       for (int j = 0; j < 4; j++) {
61          line(drawing, rect_points[j], rect_points[(j + 1) % 4], rect_color, ↩
                1, 8);
62
```

```
63          if (defects_rejection[i] == false)
64             line(image_rgb, rect_points[j], rect_points[(j + 1) % 4], ↩
                   rect_color, 2, 8);
65          else
66             line(image_rgb, rect_points[j], rect_points[(j + 1) % 4], Scalar↩
                   (0, 223, 15), 1, 8);
67       }
68
69    }
70
71    /// Calculate the area with the moments 00
72    printf("%d aproved defects)\n\n", total_defs);
73    for (int i = 0; i < contours.size(); i++)
74    {
75       if (defects_rejection[i] == false)
76          printf(" Aproved ");
77       else
78          printf(" Rejected");
79
80       printf(" * Defect[%d] - Area (M_00) = %.2f - Perimeter: %.2f - Mass ↩
              center(x, y): (%.2f, %.2f)\n", i,
81          mu[i].m00, arcLength(contours[i], true), mc[i].x, mc[i].y);
82    }
83
84    /// Show in a window
85    if (show_images) namedWindow("All the " + win_name, CV_WINDOW_AUTOSIZE);
86    if (show_images) imshow("All the " + win_name, drawing);
87
88    Mat crop_final = drawing(myROI);
89    //imwrite("contours_def_id_lat.png", crop_final);
90
91
92    /// Show in a window
93    if (show_images) namedWindow("Detected " + win_name + " - Original image",↩
          CV_WINDOW_AUTOSIZE);
94    if (show_images) imshow("Detected " + win_name + " - Original image", ↩
         image_rgb);
95
96    crop_final = image_rgb(myROI);
97    //imwrite("original_def_id_lat.png", crop_final);
98
99    /*Crops image by the mask*/
100
101   bit_part = Mat::zeros(image_rgb.rows, image_rgb.cols, image_rgb.type());
102
103   Mat mask_color;
104   cvtColor(mask, mask_color, cv::COLOR_GRAY2BGR);
105   bitwise_and(mask_color, image_rgb, bit_part);
106
```

```
107     if (true) {
108         namedWindow("Original image cropped with defects", WINDOW_AUTOSIZE); //↩
                Create a window for display.
109         imshow("Original image cropped with defects", bit_part); // Show our ↩
                image inside it.
110     }
111 
112     return total_defs;
113 }
```

C.4 Funções desenvolvidas para reconstruir a superfície superior da peça

```
1  void transform_sup_zone(Point2f srcTri[4], Mat src, Mat &dest, double r, ↩
       double ang, double angle_ajust) {
2  
3      Mat warp_mat;
4  
5      angle_ajust = angle_ajust * 3.14159265359 / 180.0;
6      ang = (ang * 3.14159265359 / 180.0) + angle_ajust;
7  
8      //system("pause");
9      dest = Mat::zeros(src.rows, src.cols, src.type());
10 
11     Point2f dstTri[4];
12     dstTri[0] = Point2f((src.cols) / 2, (src.rows) / 2);
13     dstTri[1] = Point2f(((src.cols) / 2) + r*sin(ang), ((src.rows) / 2) + r*↩
           cos(ang));
14     dstTri[2] = Point2f((src.cols) / 2, ((src.rows) / 2) + r);
15     dstTri[3] = Point2f(((src.cols) / 2) + r*sin(ang / 2), ((src.rows) / 2) + ↩
           r*cos(ang / 2));
16 
17     warp_mat = getPerspectiveTransform(srcTri, dstTri);
18 
19     warpPerspective(src, dest, warp_mat, dest.size());
20 
21     if (show_images) namedWindow("src", WINDOW_AUTOSIZE);
22     if (show_images) imshow("src", src);
23 
24     if (show_images) namedWindow("dest", WINDOW_AUTOSIZE);
25     if (show_images) imshow("dest", dest);
26 
27 }
```

C.5 Função desenvolvida para reconstruir a superfície lateral da peça

```
void transform_lat_zone(Point2f srcP[4], Mat src, Mat &dest, double width, ↩
    double heigth, double space_ajust, int step) {

   Mat warp_mat;

   dest = Mat::zeros(src.rows, src.cols, src.type());

   Point2f dstQuad[4];
   dstQuad[0] = Point2f(width*(step-1), 0);
   dstQuad[1] = Point2f(width*(step - 1) + width, 0);
   dstQuad[2] = Point2f(width*(step - 1) + width, heigth);
   dstQuad[3] = Point2f(width*(step - 1), heigth);

   warp_mat = getPerspectiveTransform(srcP, dstQuad);

   warpPerspective(src, dest, warp_mat, dest.size());

   if (show_images) namedWindow("src", WINDOW_AUTOSIZE);
   if (show_images) imshow("src", src);

   if (show_images) namedWindow("dest", WINDOW_AUTOSIZE);
   if (show_images) imshow("dest", dest);

}
```

C.6 Função desenvolvida para controlar as trajetórias do *UR5*

Nota: Foi utilizada a biblioteca para programar este manipulador apresentada anteriormente.

```
private: System::Void button19_Click(System::Object^  sender, System::↩
    EventArgs^  e) {

   demo d("Vision Inspection");
   universal_robot robot("UR5", "30002", "192.168.1.102", "TCP"); //UR5 ↩
       initialization

   universal_robot::pose p0 = d.read_pose_from_file("Inspecao_visao/↩
       pontos_trajetoria/p0.txt");
   universal_robot::pose p1 = d.read_pose_from_file("Inspecao_visao/↩
       pontos_trajetoria/p1.txt");
   universal_robot::pose p2 = d.read_pose_from_file("Inspecao_visao/↩
       pontos_trajetoria/p2.txt");

```

```
10  universal_robot::pose p3 = d.read_pose_from_file("Inspecao_visao/↩
        pontos_trajetoria/p3.txt");
11  universal_robot::pose p4 = d.read_pose_from_file("Inspecao_visao/↩
        pontos_trajetoria/p4.txt");
12
13  vector<double> p3_jp = d.read_ur_joints_pos_from_file("Inspecao_visao/↩
        pontos_trajetoria/p3_joints_pos.txt");
14
15  vector<double> p4_jp = d.read_ur_joints_pos_from_file("Inspecao_visao/↩
        pontos_trajetoria/p4_joints_pos.txt");
16
17  double p3_joints_pos[6];
18  double p4_joints_pos[6];
19
20  for (int i = 0; i < 6; i++) {
21     p3_joints_pos[i] = p3_jp[i];
22     p4_joints_pos[i] = p4_jp[i];
23  }
24
25  universal_robot::pose p_aux;
26
27  double step_sup_insp = System::Convert::ToDouble(step_sup_inspection->↩
        Value);
28  double step_lat_insp = System::Convert::ToDouble(step_lat_inspection->↩
        Value);
29  double min_area = System::Convert::ToDouble(min_area_def->Value);
30
31  double vel_rot = universal_robot::degree_to_rad(System::Convert::ToDouble(↩
        v_rot->Value) );
32  double vel_lin = System::Convert::ToDouble(v_lin->Value);
33  double acc_rot = universal_robot::degree_to_rad( System::Convert::ToDouble↩
        (a_rot->Value) );
34  double acc_lin = System::Convert::ToDouble(a_lin->Value);
35
36  p0.r = 0.01; p1.r = 0.01; p2.r = 0.01;
37  p3.r = 0.0; p4.r = 0.0;
38
39  p0.v = vel_lin; p1.v = vel_lin; p2.v = vel_lin;
40  p3.v = vel_lin; p4.v = vel_lin;
41
42  p0.a = acc_lin; p1.a = acc_lin; p2.a = acc_lin;
43  p3.a = acc_lin; p4.a = acc_lin;
44
45  universal_robot::pose tcp;
46  tcp.X = 0;
47  tcp.Y = 0;
48  tcp.Z = 0.22095;
49  tcp.Rx = 0;
50  tcp.Ry = 0;
```

```
51    tcp.Rz = 0;
52
53    double payload = 2.3;
54
55    double gravity[3] = { 9.82, 0, 0 };
56
57    double total_rotation = 0.0, p3_init_j6_pos = p3_joints_pos[5], ↩
          p4_init_j6_pos = p4_joints_pos[5];
58
59    char pos_x, pos_y;
60
61    double q_aux[6];
62
63    robot.conn.open_comm();
64
65    socket_comm vision_system_conn("TCP", "8889", "localhost", "vision_system"↩
          );
66    vision_system_conn.open_comm();
67
68    ostringstream s;
69    char *send_buffer;
70    char recv_buffer[DEFAULT_BUFFER_LEN];
71
72    s.str(std::string());
73    s << "ROBOPOLI_READY_CONFIG_" << step_sup_insp << "_" << step_lat_insp;
74
75    send_buffer = _strdup(s.str().c_str());
76    vision_system_conn.send_msg(send_buffer, strlen(send_buffer));
77
78    do {
79       vision_system_conn.recv_msg(recv_buffer, DEFAULT_BUFFER_LEN);
80    } while (strncmp("VISIONSYSTEM_READY", recv_buffer, strlen("↩
          VISIONSYSTEM_READY")) != 0);
81
82    while (1) {
83
84       //robot.send_prog_header(gravity, tcp, payload);
85       s.str(std::string());
86
87       robot.movej(false, p3_joints_pos, p0, acc_rot, vel_rot, 0.0, 0.0);
88
89       s << "READY_";
90
91       s << "P_";
92
93       cout << "çãPosio X: ";
94       cin >> pos_x;
95       cout << endl;
96
```

```
97      s << pos_x;
98
99      cout << "çãPosio Y: ";
100     cin >> pos_y;
101     cout << endl;
102
103     s << pos_y;
104
105     send_buffer = _strdup(s.str().c_str());
106     vision_system_conn.send_msg(send_buffer, strlen(send_buffer));
107
108     s << "_OK";
109
110     do {
111        vision_system_conn.recv_msg(recv_buffer, DEFAULT_BUFFER_LEN);
112     } while (strncmp(_strdup(s.str().c_str()), recv_buffer, strlen(_strdup(↩
           s.str().c_str()))) != 0);
113
114     robot.wait_for_next_pose(true, p0, q_aux);
115
116     robot.movej(false, p3_joints_pos, p1, acc_rot, vel_rot, 0.0, 0.0);
117
118     robot.wait_for_next_pose(true, p1, q_aux);
119
120     //robot.movel(p2);
121     robot.movej(false, p3_joints_pos, p2, acc_rot, vel_rot, 0.0, 0.0);
122
123     robot.wait_for_next_pose(true, p2, q_aux);
124
125     robot.movel(p3);
126
127     robot.wait_for_next_pose(true, p3, q_aux);
128
129     while (total_rotation < 360) {
130
131        robot.movej(true, p3_joints_pos, p_aux, acc_rot, vel_rot, 0.0, 0.0);
132
133        robot.wait_for_next_pose(false, p_aux, p3_joints_pos);
134
135        s.str(std::string());
136
137        s << "P_";
138
139        s << pos_x << pos_y;
140
141        s << "_sup_" << total_rotation;
142
143        send_buffer = _strdup(s.str().c_str());
144        vision_system_conn.send_msg(send_buffer, strlen(send_buffer));
```

```
145
146         s << "_OK";
147
148         do {
149            vision_system_conn.recv_msg(recv_buffer, DEFAULT_BUFFER_LEN);
150         } while (strncmp(_strdup(s.str().c_str()), recv_buffer, strlen(↩
                _strdup(s.str().c_str()))) != 0);
151
152
153
154         total_rotation += step_sup_insp;
155         p3_joints_pos[5] = p3_init_j6_pos + universal_robot::degree_to_rad(↩
                total_rotation);
156
157      }
158      total_rotation = 0;
159
160      s.str(std::string());
161
162      s << "P_PARCIAL_END_";
163
164      s << pos_x << pos_y;
165
166      s << "_sup";
167
168      send_buffer = _strdup(s.str().c_str());
169      vision_system_conn.send_msg(send_buffer, strlen(send_buffer));
170
171      s << "_OK";
172
173      do {
174         vision_system_conn.recv_msg(recv_buffer, DEFAULT_BUFFER_LEN);
175      } while (strncmp(_strdup(s.str().c_str()), recv_buffer, strlen(_strdup(↩
             s.str().c_str()))) != 0);
176
177      p3_joints_pos[5] = p3_init_j6_pos;
178      robot.movej(true, p3_joints_pos, p_aux, acc_rot, vel_rot, 0.0, 0.0);
179
180      robot.wait_for_next_pose(false, p_aux, p3_joints_pos);
181
182      robot.movel(p4);
183
184      robot.wait_for_next_pose(true, p4, q_aux);
185
186      while (total_rotation < 360) {
187
188         robot.movej(true, p4_joints_pos, p_aux, acc_rot, vel_rot, 0.0, 0.0);
189
190         robot.wait_for_next_pose(false, p_aux, p4_joints_pos);
```

```
191
192
193         s.str(std::string());
194         s << "P_";
195
196         s << pos_x << pos_y;
197
198         s << "_lat_" << total_rotation;
199
200         send_buffer = _strdup(s.str().c_str());
201         vision_system_conn.send_msg(send_buffer, strlen(send_buffer));
202
203         s << "_OK";
204
205         do {
206            vision_system_conn.recv_msg(recv_buffer, DEFAULT_BUFFER_LEN);
207         } while (strncmp(_strdup(s.str().c_str()), recv_buffer, strlen(↩
               _strdup(s.str().c_str()))) != 0);
208
209
210         total_rotation += step_lat_insp;
211         p4_joints_pos[5] = p4_init_j6_pos + universal_robot::degree_to_rad(↩
               total_rotation);
212
213      }
214      total_rotation = 0;
215
216      s.str(std::string());
217      s << "P_PARCIAL_END_";
218
219      s << pos_x << pos_y;
220
221      s << "_lat";
222
223      send_buffer = _strdup(s.str().c_str());
224      vision_system_conn.send_msg(send_buffer, strlen(send_buffer));
225
226      s << "_OK";
227
228      do {
229         vision_system_conn.recv_msg(recv_buffer, DEFAULT_BUFFER_LEN);
230      } while (strncmp(_strdup(s.str().c_str()), recv_buffer, strlen(_strdup(↩
            s.str().c_str()))) != 0);
231
232      s.str(std::string());
233
234      s << "END_";
235
236      s << "P_";
```

```
237
238         s << pos_x << pos_y;
239
240         send_buffer = _strdup(s.str().c_str());
241         vision_system_conn.send_msg(send_buffer, strlen(send_buffer));
242
243         s << "_OK";
244
245         do {
246            vision_system_conn.recv_msg(recv_buffer, DEFAULT_BUFFER_LEN);
247         } while (strncmp(_strdup(s.str().c_str()), recv_buffer, strlen(_strdup(↩
               s.str().c_str()))) != 0);
248
249
250         p4_joints_pos[5] = p4_init_j6_pos;
251         robot.movej(true, p4_joints_pos, p_aux, acc_rot, vel_rot, 0.0, 0.0);
252
253         robot.wait_for_next_pose(false, p_aux, p4_joints_pos);
254
255         robot.movel(p2);
256
257         //robot.wait_for_next_pose(true, p2, q_aux);
258         //system("pause");
259         Sleep(1500);
260
261         robot.movel(p1);
262
263         //robot.wait_for_next_pose(true, p1, q_aux);
264         Sleep(1500);
265
266         //robot.send_prog_footer();
267
268         //system("pause");
269
270      }
271
272      robot.conn.close_comm();
273
274   }
```

C.7 Função desenvolvida para adquirir as imagens de forma sincronizada com as trajetórias do *UR5*

```
1  int main(int, char**)
2  {
```

```
3
4   char recv_buffer[DEFAULT_BUFLEN];
5   int n_bytes = 0, pos, n_def_lap=0, n_def_grad = 0, n_def_lap_total = 0, ↩
        n_def_grad_total = 0;
6   char *send_buffer;
7   ostringstream s;
8   int system_ready = 0;
9   string path = std::string();
10  string delimiter = "_";
11  string s_copy, token, zone, p_number;
12  double angle, step_sup, step_lat;
13
14  VideoCapture cap(2); // open the camera
15  if (!cap.isOpened()) { // check if we succeeded
16     cout << "Error openning camera" << endl;
17     system("pause");
18     return -1;
19  }
20  cap.set(CV_CAP_PROP_FPS, 30); // set 30 frames per second
21
22  //cap.set(CV_CAP_PROP_SETTINGS, 0); // open camera settings dialog
23
24  Mat frame;
25  string last_window_name = std::string();
26
27  init_server();
28
29  //read and send data to the clint
30  while (readline(ClientSocket, recv_buffer, n_bytes)) {
31
32     if (system_ready == 0) {
33
34        if (strncmp("ROBOPOLI_READY", recv_buffer, strlen("ROBOPOLI_READY"))↩
               == 0) {
35
36           s.str(std::string());
37           s.str(std::string(recv_buffer, 0, n_bytes));
38
39           s_copy = s.str();
40
41           pos = s_copy.find(delimiter);
42           s_copy.erase(0, pos + delimiter.length());
43
44           pos = s_copy.find(delimiter);
45           s_copy.erase(0, pos + delimiter.length());
46
47           pos = s_copy.find(delimiter);
48           s_copy.erase(0, pos + delimiter.length());
49
```

```
50            pos = s_copy.find(delimiter);
51            token = s_copy.substr(0, pos);
52            step_sup = stod(token);
53            s_copy.erase(0, pos + delimiter.length());
54
55            step_lat = stod(s_copy);
56
57            std::ofstream fs;
58            fs.open("info_pecas/ensaio2/config.txt");
59            if (!fs)
60            {
61               std::cerr << "Cannot open the output file: " << "↩
                      Inspecao_visao/config.txt" << std::endl;
62               system("pause");
63            }
64
65            fs << step_sup << endl;
66            fs << step_lat;
67
68            fs.close();
69
70            s.str(std::string());
71            s << "VISIONSYSTEM_READY";
72
73            send_buffer = _strdup(s.str().c_str());
74
75            sendline(ClientSocket, send_buffer);
76
77            system_ready = 1;
78         }
79
80      }
81
82      else {
83
84         if (strncmp("READY_", recv_buffer, strlen("READY_")) == 0) {
85
86            s.str(std::string());
87            s.str(std::string(recv_buffer, 0, n_bytes));
88
89            s.str(std::string(recv_buffer, 0, n_bytes).append("_OK"));
90
91            send_buffer = _strdup(s.str().c_str());
92
93            sendline(ClientSocket, send_buffer);
94
95         }
96
97         else if (strncmp("END_", recv_buffer, strlen("END_")) == 0) {
```

```
98
99              s.str(std::string());
100             s.str(std::string(recv_buffer, 0, n_bytes));
101
102             s.str(std::string(recv_buffer, 0, n_bytes).append("_OK"));
103
104             send_buffer = _strdup(s.str().c_str());
105
106             sendline(ClientSocket, send_buffer);
107
108         }
109
110         else if (strncmp("P_", recv_buffer, strlen("P_")) == 0) {
111
112             s.str(std::string());
113             s.str(std::string(recv_buffer, 0, n_bytes));
114
115             s_copy = s.str();
116
117             cap >> frame; // get a new frame from camera
118             cap >> frame; // get a new frame from camera
119
120             if (!last_window_name.empty())
121                 destroyWindow(last_window_name);
122
123             imshow(s.str(), frame);
124             waitKey(30);
125             last_window_name = s.str();
126
127             destroyAllWindows();
128
129             pos = s_copy.find(delimiter);
130             s_copy.erase(0, pos + delimiter.length());
131
132             pos = s_copy.find(delimiter);
133             token = s_copy.substr(0, pos);
134             p_number = token;
135             s_copy.erase(0, pos + delimiter.length());
136
137             pos = s_copy.find(delimiter);
138             token = s_copy.substr(0, pos);
139             zone = token;
140             s_copy.erase(0, pos + delimiter.length());
141
142             angle = stod(s_copy);
143
144
145             path = std::string("info_pecas/ensaio2/");
146
```

```
147             path.append(p_number); path.append("/");
148 
149             if (zone.compare("sup")==0)
150                path.append("superior/");
151             else
152                path.append("lateral/");
153 
154             path.append(to_string(angle)); path.append(".png");
155 
156             cout << "image saved: " << path << endl;
157 
158             imwrite(path, frame);
159 
160 
161             /*vision_inspection(path, zone, n_def_lap, n_def_grad);
162             n_def_grad_total += n_def_grad;
163             n_def_lap_total += n_def_lap;*/
164 
165 
166             s.str(std::string(recv_buffer, 0, n_bytes).append("_OK"));
167 
168             send_buffer = _strdup(s.str().c_str());
169 
170             sendline(ClientSocket, send_buffer);
171 
172          }
173 
174          else if (strncmp("P_PARCIAL_END_", recv_buffer, strlen("↩
                P_PARCIAL_END_")) == 0) {
175 
176             s.str(std::string(recv_buffer, 0, n_bytes).append("_OK"));
177 
178             send_buffer = _strdup(s.str().c_str());
179 
180             sendline(ClientSocket, send_buffer);
181 
182          }
183 
184       }
185 
186    }
187 
188    close_server();
189 
190    return 0;
191 }
```

C.8 Função desenvolvida para processar as imagens adquiridas

```
int main(int, char**)
{
   Point2f srcPsup[4];
   srcPsup[0] = Point2f(477, 457);
   srcPsup[1] = Point2f(885, 385);
   srcPsup[2] = Point2f(897, 486);
   srcPsup[3] = Point2f(897, 432);

   Point2f srcPlat[4];
   srcPlat[0] = Point2f(362, 540);
   srcPlat[1] = Point2f(310, 474);
   srcPlat[2] = Point2f(456, 268);
   srcPlat[3] = Point2f(498, 332);

   Point2f destPlat[4];
   int width = 40;
   int heigth = width * 2;

   int max_area_grad_sup = 0, max_area_grad_lat = 0;

   int frames_lat = 0;

   double r;

   string path, path_copy, path_copy2, path_copy3;

   int n_def_grad = 0, total_def_grad = 0, total_lat_def_grad = 0, ↩
       total_sup_def_grad =0;

   path = "C:/Users/nunoj/Desktop/inspecao_pintadas/inspecao_pintadas/↩
       info_pecas/ensaio1/";

   path_copy = path;
   path_copy2 = path;

   int pos_x, pos_y;

   ifstream f;
   string line;

   path_copy.append("config.txt");
   f.open(path_copy, ifstream::in);

   if (!f.is_open())
      cout << "Error opening file: config" << endl;
```

```
44
45    getline(f, line);
46
47    double step_sup = stod(line);
48
49    getline(f, line);
50
51    double step_lat = stod(line);
52
53    f.close();
54
55    pos_y = 0; pos_x = 0;
56
57    while ( 1 ) {
58
59       sup_reconstraction_grad = Mat::zeros(768, 1024, 16);
60       lat_reconstraction_grad = Mat::zeros(768, 1024, 16);
61
62       max_area_grad_sup = 0, max_area_grad_lat = 0;
63
64       path = path_copy2;
65
66       cout << " * Posicao da peca a inspecionar: " << endl;
67       cout << "X: "; cin >> pos_x;;
68       cout << "Y: "; cin >> pos_y; cout << endl;
69
70
71
72       path.append(to_string(pos_x)); path.append(to_string(pos_y));
73
74       path_copy3 = path;
75
76       path.append( "/superior/");
77       path_copy = path;
78
79       total_def_grad = 0;
80       total_sup_def_grad = 0;
81       total_lat_def_grad = 0;
82
83       MIN_AREA_DEF = 60;
84       for (int i = 1; i <= 360 / step_sup; i++) {
85
86          path = path_copy;
87
88          path.append(to_string(360 - step_sup * i));
89          path.append(".png");
90
91          int max_area_grad_i;
92          vision_inspection(path, "sup", n_def_grad, max_area_grad_i);
```

```
93
94          if (max_area_grad_i > max_area_grad_sup)
95             max_area_grad_sup = max_area_grad_i;
96
97          total_sup_def_grad += n_def_grad;
98
99          Mat dest_affine_grad;
100         r= bit_part.rows / 2 - 1 - 100;
101
102         transform_sup_zone(srcPsup, bit_part_grad, dest_affine_grad, ↩
                bit_part.rows / 2 - 1 - 100, step_sup, -0.2);
103
104         Point2f center(bit_part.cols / 2, bit_part.rows / 2);
105
106         Mat dest_rot_grad;
107
108         rotate(dest_affine_grad, step_sup * i, dest_rot_grad, center);
109
110         bitwise_or(dest_rot_grad, sup_reconstraction_grad, ↩
                sup_reconstraction_grad);
111
112         if (show_images) waitKey(0);
113
114         if (show_images) destroyAllWindows();
115      }
116
117
118      path = path_copy3;
119
120      path.append("/lateral/");
121      path_copy = path;
122
123      MIN_AREA_DEF = 70;
124      for (int i = 1; i <= 360 / step_lat; i++) {
125
126         path = path_copy;
127
128         path.append(to_string(360 - step_lat*i));
129         path.append(".png");
130
131         int max_area_grad_i;
132         vision_inspection(path, "lat", n_def_grad, max_area_grad_i);
133
134         if (max_area_grad_i > max_area_grad_lat)
135            max_area_grad_lat = max_area_grad_i;
136
137         total_lat_def_grad += n_def_grad;
138
139         Mat dest_affine_lap, dest_affine_grad;
```

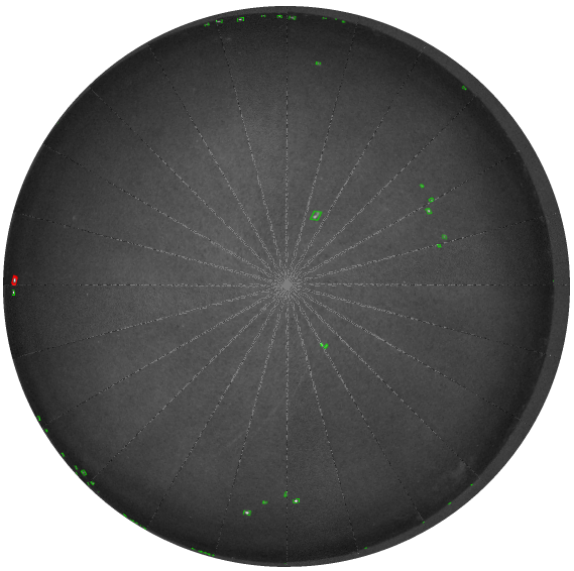
```
140         transform_lat_zone(srcPlat, bit_part_grad, dest_affine_grad, width, ↩
                heigth, 0, i);
141
142         bitwise_or(dest_affine_grad, lat_reconstraction_grad, ↩
                lat_reconstraction_grad);
143
144         if (show_images) waitKey(0);
145
146         if (show_images) destroyAllWindows();
147
148         frames_lat = i;
149      }
150
151      total_def_grad = total_lat_def_grad + total_sup_def_grad;
152
153      cout << " ** Resumo ** " << endl;
154      cout << " - Gradiente" << endl;
155      cout << "Total de defeitos parte superior: " << total_sup_def_grad << ↩
            endl;
156      cout << "Total de defeitos parte lateral: " << total_lat_def_grad << ↩
            endl;
157      cout << "Total: " << total_def_grad << endl;
158
159      cout << " * Areas maximas:" << endl;
160      cout << " - Gradiente" << endl;
161      cout << "Superior: " << max_area_grad_sup << endl;
162      cout << "Lateral: " << max_area_grad_lat << endl;
163
164      cout << "sup_max_time: " << sup_max_time << endl;
165      cout << "lat_max_time: " << lat_max_time << endl;
166      cout << "load_time: " << load_time << endl;
167
168      // Setup a rectangle to define your region of interest
169      cv::Rect myROI_sup(225, 100, 2.02 * r, 2.02* r);
170
171      // Crop the full image to that image contained by the rectangle myROI
172      sup_reconstraction_grad = sup_reconstraction_grad(myROI_sup);
173      namedWindow("Reconstucao da parte superior da peca - Morphological ↩
            gradient", WINDOW_AUTOSIZE);
174      imshow("Reconstucao da parte superior da peca - Morphological gradient"↩
            , sup_reconstraction_grad);
175
176      string s_aux = "resultados finais/sup_";
177      s_aux.append(to_string(pos_x));
178      s_aux.append(to_string(pos_y));
179      s_aux.append(".png");
180
181
182      imwrite(s_aux, sup_reconstraction_grad);
```

```
183
184         // Setup a rectangle to define your region of interest
185         cv::Rect myROI(0, 0, width*frames_lat, heigth);
186
187         // Crop the full image to that image contained by the rectangle myROI
188         lat_reconstraction_grad = lat_reconstraction_grad(myROI);
189
190         namedWindow("Reconstucao da parte lateral da peca - Morphological ↩
                gradient", WINDOW_AUTOSIZE);
191         imshow("Reconstucao da parte lateral da peca - Morphological gradient",↩
                 lat_reconstraction_grad);
192
193         s_aux = "resultados finais/lat_";
194         s_aux.append(to_string(pos_x));
195         s_aux.append(to_string(pos_y));
196         s_aux.append(".png");
197
198         imwrite(s_aux, lat_reconstraction_grad);
199
200         waitKey(20);
201         destroyAllWindows();
202     }
203
204 }
```

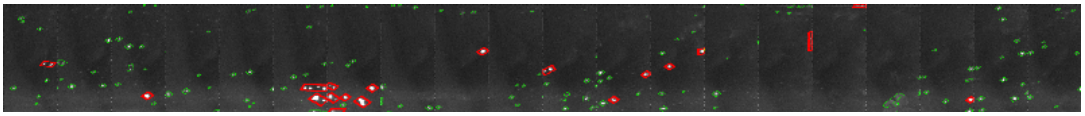
Anexo D

Inspeção por visão artificial de peças altamente refletoras e espelhadas - Resultados finais

D.1 Peças boas

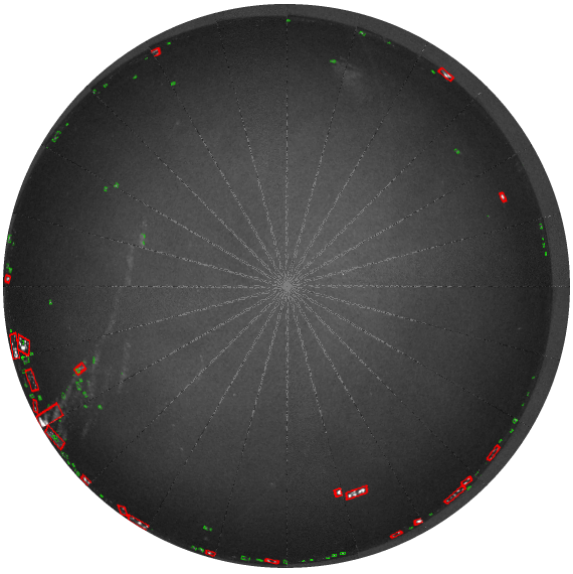


(a) Superfície superior

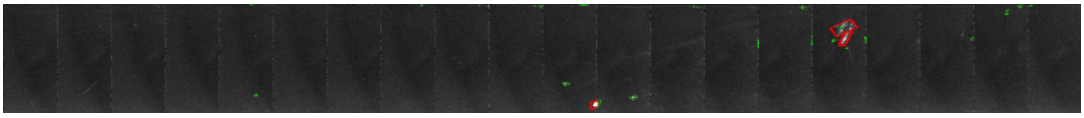


(b) Superfície lateral

Figura D.1: Resultados finais obtidos - Peça 00

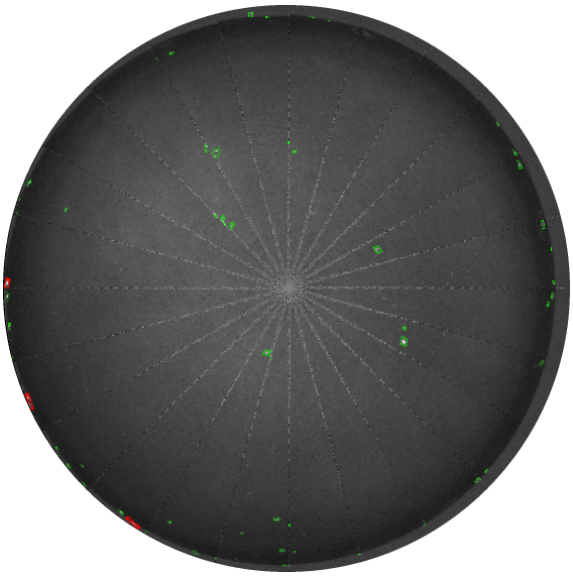


(a) Superfície superior

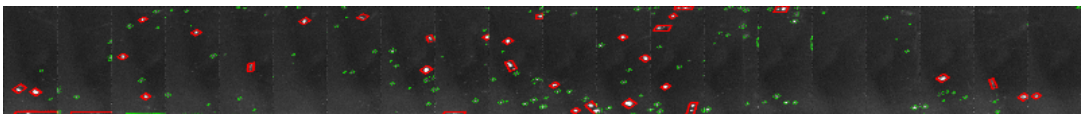


(b) Superfície lateral

Figura D.2: Resultados finais obtidos - Peça 01

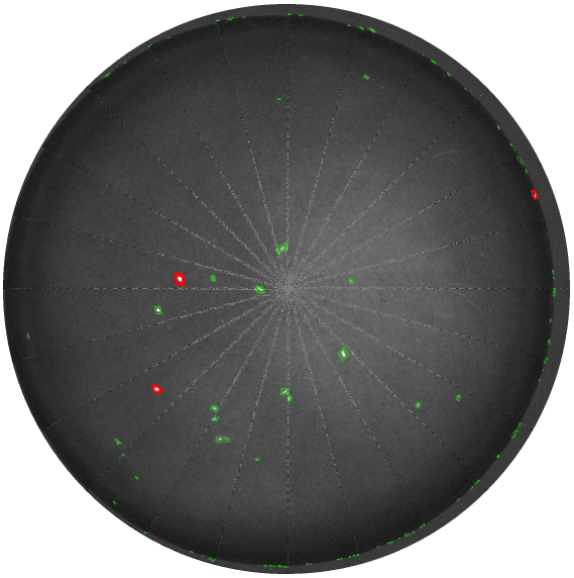


(a) Superfície superior

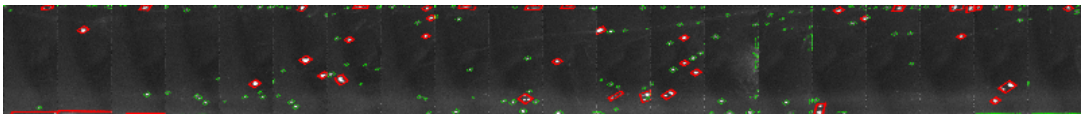


(b) Superfície lateral

Figura D.3: Resultados finais obtidos - Peça 02

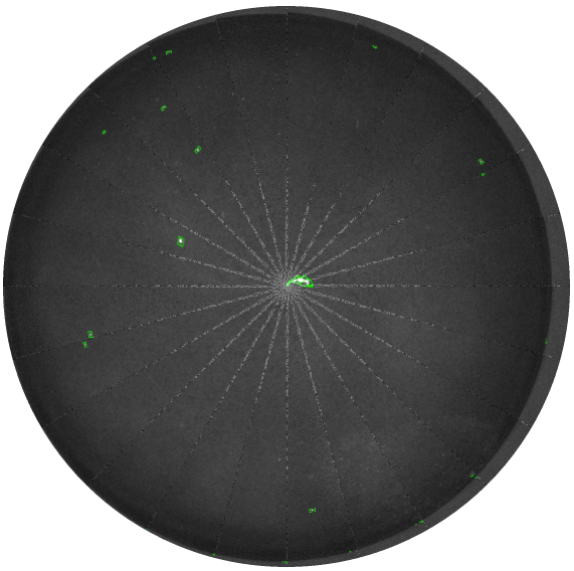


(a) Superfície superior

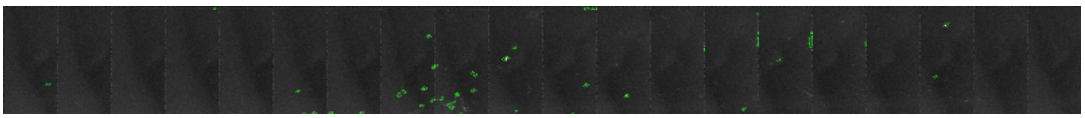


(b) Superfície lateral

Figura D.4: Resultados finais obtidos - Peça 03

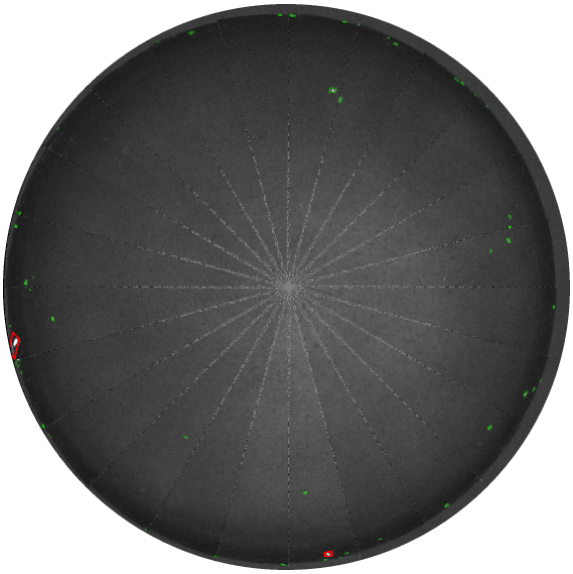


(a) Superfície superior

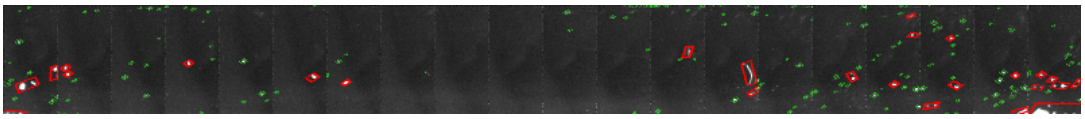


(b) Superfície lateral

Figura D.5: Resultados finais obtidos - Peça 10

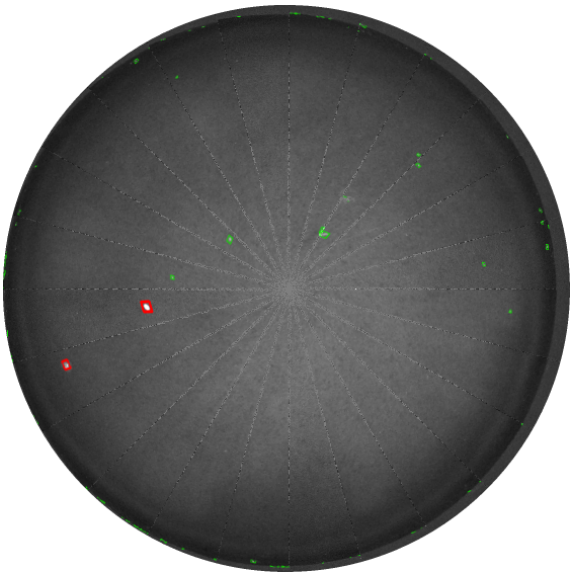


(a) Superfície superior

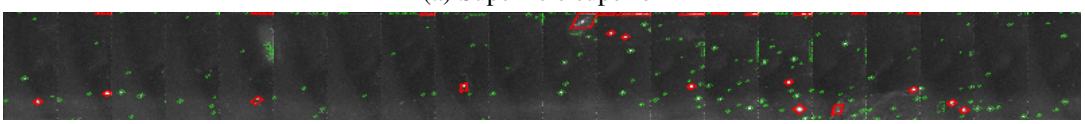


(b) Superfície lateral

Figura D.6: Resultados finais obtidos - Peça 11

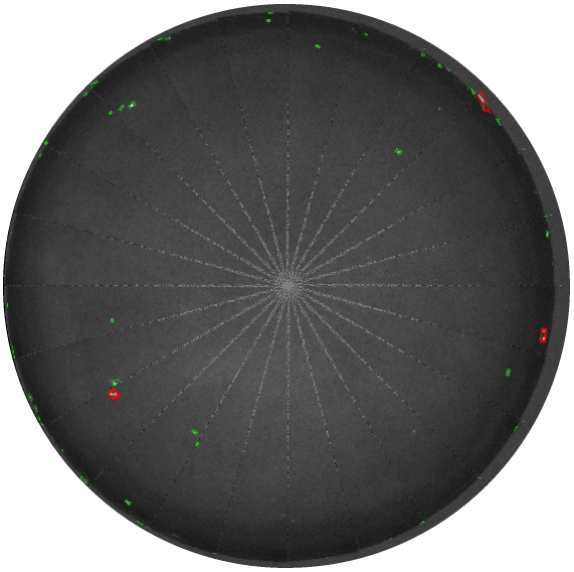


(a) Superfície superior

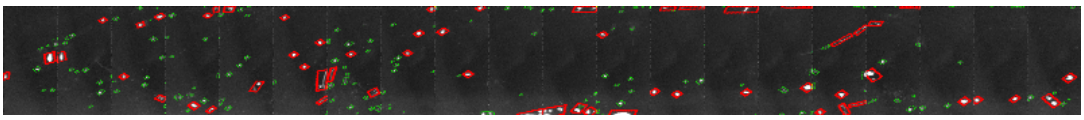


(b) Superfície lateral

Figura D.7: Resultados finais obtidos - Peça 12



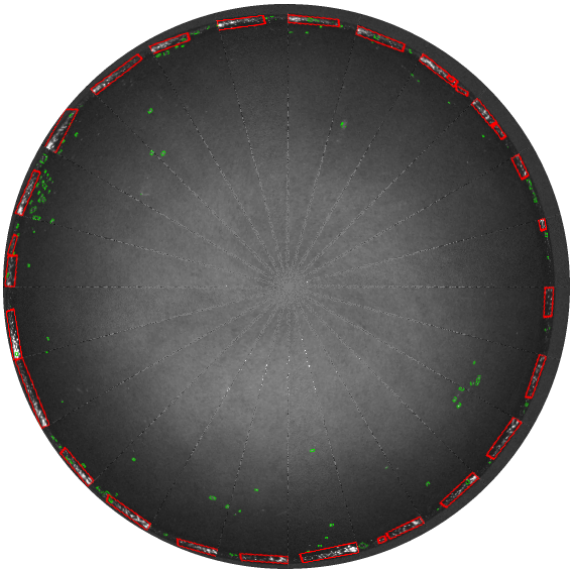
(a) Superfície superior



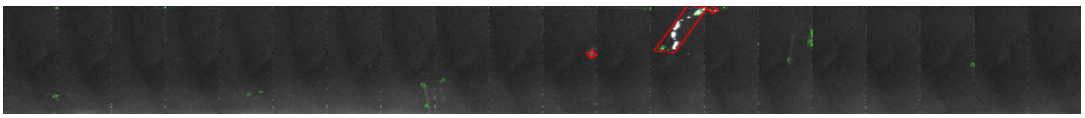
(b) Superfície lateral

Figura D.8: Resultados finais obtidos - Peça 13

D.2 Peças más

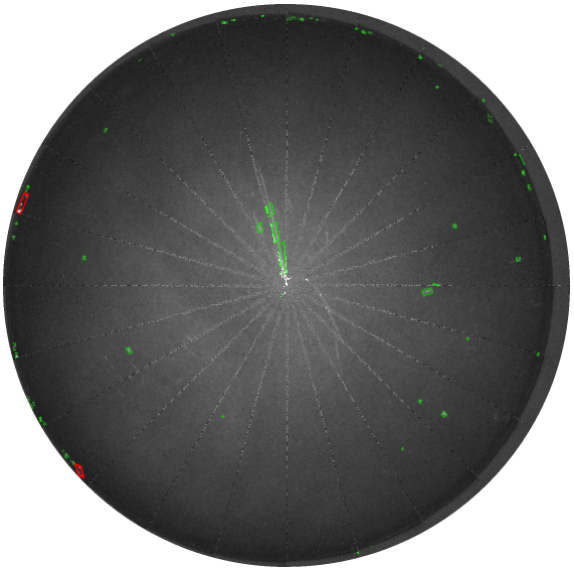


(a) Superfície superior

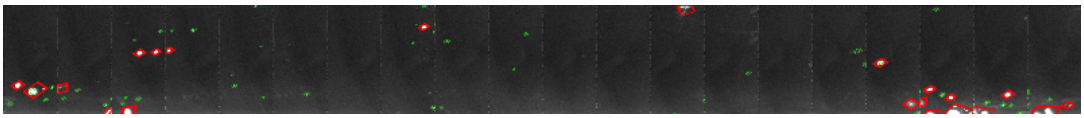


(b) Superfície lateral

Figura D.9: Resultados finais obtidos - Peça 31

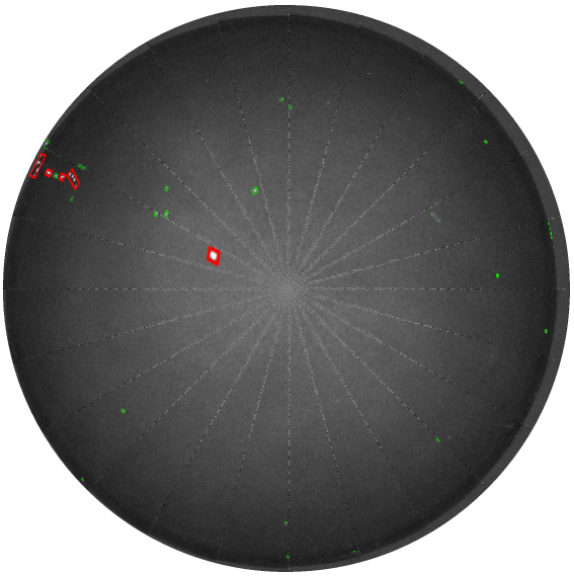


(a) Superfície superior

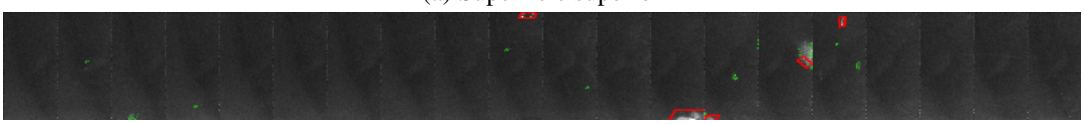


(b) Superfície lateral

Figura D.10: Resultados finais obtidos - Peça 32

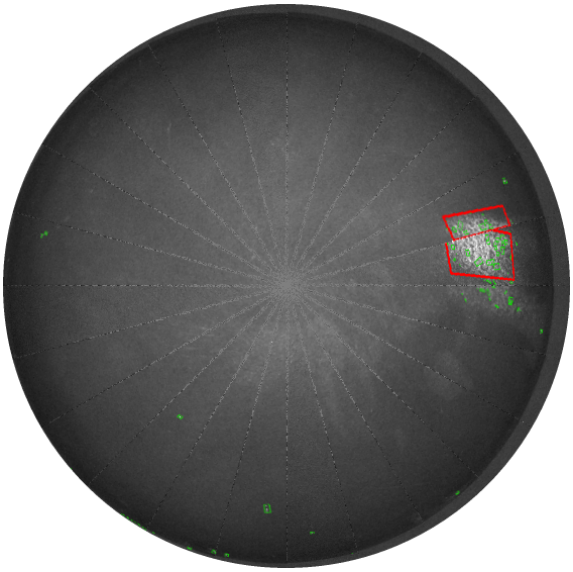


(a) Superfície superior

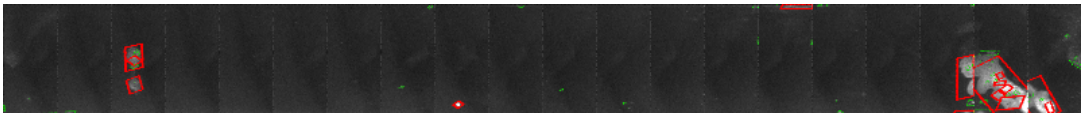


(b) Superfície lateral

Figura D.11: Resultados finais obtidos - Peça 33

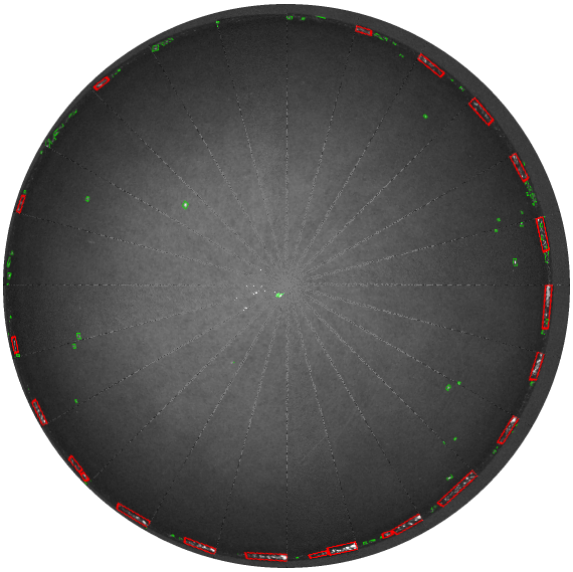


(a) Superfície superior

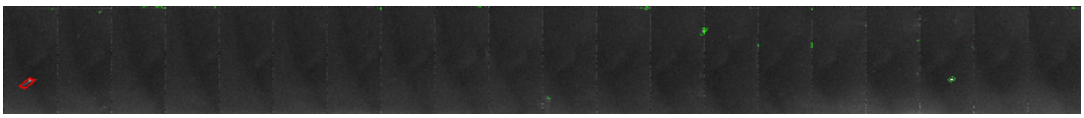


(b) Superfície lateral

Figura D.12: Resultados finais obtidos - Peça 40

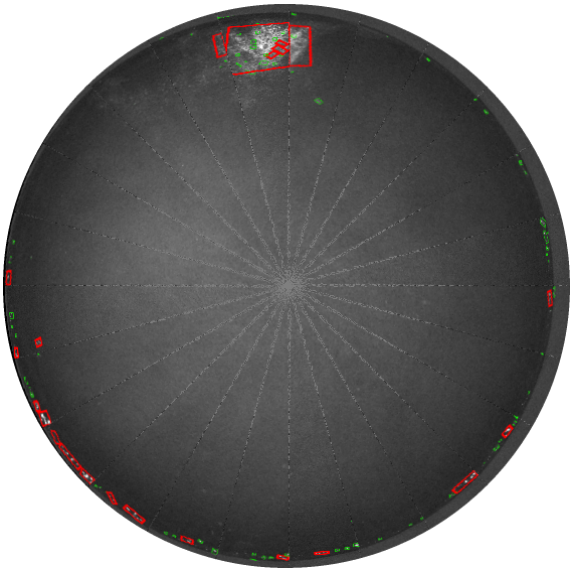


(a) Superfície superior

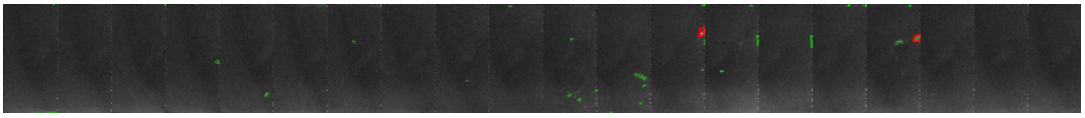


(b) Superfície lateral

Figura D.13: Resultados finais obtidos - Peça 41

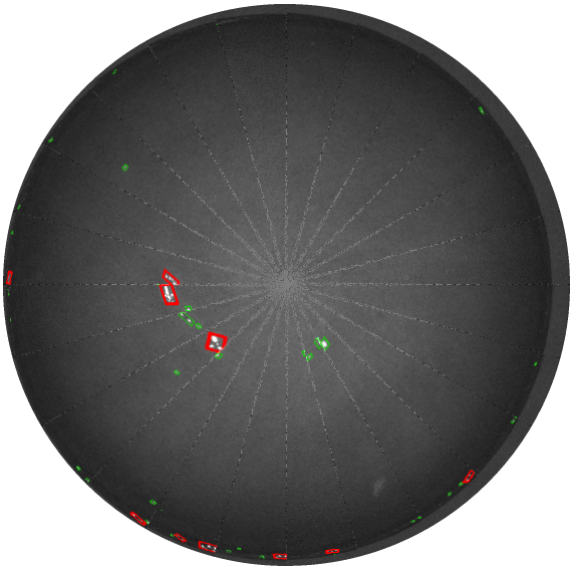


(a) Superfície superior

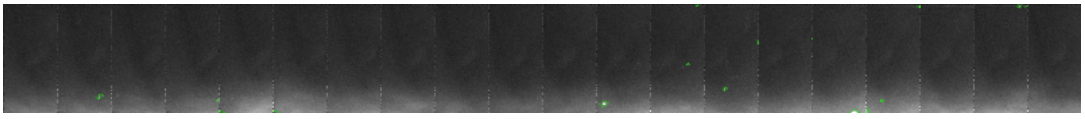


(b) Superfície lateral

Figura D.14: Resultados finais obtidos - Peça 42



(a) Superfície superior



(b) Superfície lateral

Figura D.15: Resultados finais obtidos - Peça 43

Referências

[1] Wang Jianjun, G. Zhang, Zhang Hui, e T. Fuhlbrigge. Force control technologies for new robotic applications. Em *2008 IEEE International Conference on Technologies for Practical Robot Applications (TePRA), 10-11 Nov. 2008*, 2008 IEEE International Conference on Technologies for Practical Robot Applications (TePRA), páginas 143–9. IEEE. 10408556 force control technology robotics industry assembly polishing deburring milling. URL: http://dx.doi.org/10.1109/TEPRA.2008.4686689, doi:10.1109/TEPRA.2008.4686689.

[2] Joaquim Norberto Cardoso Pires da Silva. *Realização de Controlo de Força em Robôs Manipuladores Industriais*. Thesis, 1999.

[3] Abílio Joaquim Gomes de Oliveira Azenha. *Análise dinâmica e controlo de forças em manipuladores robóticos Documento electrónico*. FEUP, Porto:, 1998. Abílio Joaquim Gomes de Oliveira Azenha. URL: http://eos.fe.up.pt:1801/webclient/DeliveryManager?custom_att_2=simple_viewer&metadata_request=false&pid=2102.

[4] Marcos André Magalhães Ferreira. *High level programmable and flexible industrial robotized cells Recurso eletrónico*. FEUP, Porto:, 2014. Bibliografia: p. 135-143 Marcos Ferreira. URL: http://digitool.fe.up.pt:1801/webclient/DeliveryManager?custom_att_2=simple_viewer&metadata_request=false&pid=744348.

[5] A. M. Mendonça. Sistemas baseados em visão - slides de condicionamento de cena. 2014.

[6] Yongtae Do, Sangok Lee, e Yoonsu Kim. Vision-based surface defect inspection of metal balls. *Measurement Science and Technology*, 22(10), 2011. Compilation and indexing terms, Copyright 2015 Elsevier Inc. 20113814337569 Ball roll Defect inspection Gray-scale images Machine vision systems metal ball Metal surfaces Reference image Spatial resolution Specular reflectance Vision based. URL: http://dx.doi.org/10.1088/0957-0233/22/10/107001, doi:10.1088/0957-0233/22/10/107001.

[7] Zhong Wang, Qian Xing, Luhua Fu, e Hong Sun. Realtime vision-based surface defect inspection of steel balls. *Transactions of Tianjin University*, 21(1):76–82, 2015. Compilation and indexing terms, Copyright 2015 Elsevier Inc. 20150900566410 Defect inspection Diffuse illumination Geometrical analysis Image processing and analysis Precision and recall Realtime inspection Steel balls Surface defect inspections. URL: http://dx.doi.org/10.1007/s12209-015-2452-6, doi:10.1007/s12209-015-2452-6.

[8] Roberto José Magalhães da Silva. *Localização de AGVs industriais baseada em marcadores*. [s. n.], Porto:, 2011. Bibliografia: p. 139-142 Roberto José Magalhães da Silva. URL: http://digitool.fe.up.pt:1801/webclient/DeliveryManager?custom_att_2=simple_viewer&metadata_request=false&pid=239144.

[9] Chuang Li, Zhenhua Wang, Cheng Fan, Guodong Chen, e Ting Huang. Research on grinding and polishing force control of compliant flange. Em *International Conference on Engineering Technology and Application, ICETA 2015, May 29, 2015 - May 30, 2015*, volume 22 de *MATEC Web of Conferences*. EDP Sciences. Compilation and indexing terms, Copyright 2016 Elsevier Inc. 20154201387712 Air springs Compliance Control strategies Fuzzy - pid controls Grinding and polishing Production efficiency Proportional valves Real time requirement. URL: http://dx.doi.org/10.1051/matecconf/20152203012, doi:10.1051/matecconf/20152203012.

[10] C. Brecher, R. Tuecks, R. Zunke, e C. Wenzel. Development of a force controlled orbital polishing head for free form surface finishing. *Production Engineering*, 4(2-3):269–77, 2010. 11949119 force controlled orbital polishing head free form surface finishing surface roughness steel mold production optics manufacturing automated polishing cell reproducible finishing process six-axis industrial robot CAD/CAM software. URL: http://dx.doi.org/10.1007/s11740-010-0221-x, doi:10.1007/s11740-010-0221-x.

[11] Fusaomi Nagata, Yukihiro Kusumoto, Yoshihiro Fujimoto, e Keigo Watanabe. Robotic sanding system for new designed furniture with free-formed surface. *Robotics and Computer-Integrated Manufacturing*, 23(4):371–379, 2007. Compilation and indexing terms, Copyright 2016 Elsevier Inc. 20071310518763 Cutter location data Non taught operations Polishing forces Robotic sanding system Surface following control. URL: http://dx.doi.org/10.1016/j.rcim.2006.04.004, doi:10.1016/j.rcim.2006.04.004.

[12] F. Nagata, T. Hase, Z. Haga, M. Omoto, e K. Watanabe. Cad/cam-based position/force controller for a mold polishing robot. *Mechatronics*, 17(4-5):207–16, 2007. 9504073 CAD/CAM position controller force controller mold polishing robot force feedback loop Cartesian space cutter location data industrial robot. URL: http://dx.doi.org/10.1016/j.mechatronics.2007.01.003, doi:10.1016/j.mechatronics.2007.01.003.

[13] Fusaomi Nagata, Takanori Mizobuchi, Shintaro Tani, Keigo Watanabe, Tetsuo Hase, e Zenku Haga. Impedance model force control using a neural network-based effective stiffness estimator for a desktop nc machine tool. *Journal of Manufacturing Systems*, 28(2-3):78–87, 2009. Compilation and indexing terms, Copyright 2016 Elsevier Inc. 20101712878694 Abrasive tools Compliance control Critical damping Effective stiffness Finishing process Force controller Force sensor High quality Impedance models Lens mold Manufacturing industries Metallic mold NC machine tools Non-Linearity Tuning method Work pieces. URL: http://dx.doi.org/10.1016/j.jmsy.2010.02.001, doi:10.1016/j.jmsy.2010.02.001.

[14] N. Mendes, P. Neto, J. Norberto Pires, e A. Paulo Moreira. Fuzzy-pi force control for industrial robotics. Em *Communications in Computer and Information Science*, volume 103 CCIS, páginas 322–329. 2010. Cited By :2 Export Date: 30 January 2016. URL: http://www.scopus.com/inward/record.url?eid=2-s2.0-78149274845&partnerID=40&md5=aef69b3a24d9f799c09764575e42c891, doi:10.1007/978-3-642-15810-0 41.

[15] Hsu Feng-Yi e Fu Li-Chen. Intelligent robot deburring using adaptive fuzzy hybrid position/force control. *IEEE Transactions on Robotics and Automation*, 16(4):325–35, 2000. 6701195 intelligent robot deburring adaptive fuzzy hybrid position/force control imprecise knowledge adaptive fuzzy control approach control architecture outer-loop command generator chamfer depth B-spline type membership functions global stability industrial robot arm. URL: http://dx.doi.org/10.1109/70.864223, doi:10.1109/70.864223.

[16] Hélder Miguel Tavares Moreira. *Ensinamento rápido de manipuladores industriais e controlo de força em operações de polimento*. [s. n.], Porto:, 2012. Bibliografia: f. 73 - 75 Hélder Miguel Tavares Moreira. URL: http://digitool.fe.up.pt:1801/webclient/DeliveryManager?custom_att_2=simple_viewer&metadata_request=false&pid=247768.

[17] F. Nagata, K. Watanabe, S. Hashino, H. Tanaka, T. Matsuyama, e K. Hara. Polishing robot using a joystick controlled teaching system. Em *Proceedings of 2000 IEEE International Conference on Industrial Electronics, Control and Instrumentation, 22-28 Oct. 2000*, volume vol.1 de *2000 26th Annual Conference of the IEEE Industrial Electronics Society. IECON 2000. 2000 IEEE International Conference on Industrial Electronics, Control and Instrumentation. 21st Century Technologies and Industrial Opportunities (Cat. No.00CH37141)*, páginas 632–7. IEEE. 7230610 industrial robot polishing robot joystick controlled teaching system impedance model following control orientation compensation force control contact force open architecture controller. URL: http://dx.doi.org/10.1109/IECON.2000.973223, doi:10.1109/IECON.2000.973223.

[18] Choi Myoung Hwan e Lee Woo Won. A force/moment sensor for intuitive robot teaching application. Em *2001 IEEE International Conference on Robotics and Automation (ICRA), May 21, 2001 - May 26, 2001*, volume 4 de *Proceedings - IEEE International Conference on Robotics and Automation*, páginas 4011–4016. Institute of Electrical and Electronics Engineers Inc. Compilation and indexing terms, Copyright 2016 Elsevier Inc. 2001416683989 Robot teaching. URL: http://dx.doi.org/10.1109/ROBOT.2001.933244, doi:10.1109/ROBOT.2001.933244.

[19] F. Nagata, S. Yoshitake, A. Otsuka, K. Watanabe, e M. K. Habib. Automatic control of an orthogonal-type robot with a force sensor and a small force input device. Em *IECON 2011 - 37th Annual Conference of IEEE Industrial Electronics, 7-10 Nov. 2011*, IECON 2011 - 37th Annual Conference of IEEE Industrial Electronics, páginas 3270–5. IEEE. 12537654 orthogonal-type robot automatic control system force sensor small force input device force feedback loop position feedback loop position feedforward loop kinetic friction forces stability criterion force control system pick feed direction cutter location data fuzzy reasoning stick-slip motion control strategy cooperative motion position control. URL: http://dx.doi.org/10.1109/IECON.2011.6119835, doi:10.1109/IECON.2011.6119835.

[20] P. Malheiros, P. Costa, e A. Paulo Moreira. 3d object motion tracking and locating system by means of synchronized light emitters with a stereoscopic vision system, 2010-04-29 2010. URL: http://worldwide.espacenet.com/publicationDetails/biblio?FT=D&date=20100429&DB=worldwide.espacenet.com&locale=en_EP&CC=WO&NR=2010046759A2&KC=A2&ND=4.

[21] Chiu-Chun Ngan e Hon-Yuen Tam. A non-contact technique for the on-site inspection of molds and dies polishing. *Journal of Materials Processing Technology*, 155-156(1-3):1184–1188, 2004. Compilation and indexing terms, Copyright 2016 Elsevier Inc. 2004518730795 Metallic surface inspection Molds and dies polishing Robot-assisted polishing. URL: http://dx.doi.org/10.1016/j.jmatprotec.2004.04.263, doi:10.1016/j.jmatprotec.2004.04.263.

[22] Adnan Rachmat Anom Besari, Anton Satria Prabuwono, Ruzaidi Zamri, e Md Dan Md Palil. Computer vision approach for robotic polishing application using artificial neural

networks. Em *2010 8th IEEE Student Conference on Research and Development - Engineering: Innovation and Beyond, SCOReD 2010, December 13, 2010 - December 14, 2010*, Proceeding, 2010 IEEE Student Conference on Research and Development - Engineering: Innovation and Beyond, SCOReD 2010, páginas 281–286. IEEE Computer Society. Compilation and indexing terms, Copyright 2016 Elsevier Inc. 20110913706060 Artificial Neural Network Automation systems Computer vision techniques Further development Manufacturing process Manufacturing time Polishing processs Polishing robots Polishing time Quality inspection Repetitive works Robotic polishing Robotics systems Surface data Surface parameter System study Vision systems Vision-based methods Working conditions. URL: http://dx.doi.org/10.1109/SCORED.2010.5704017, doi:10.1109/SCORED.2010.5704017.

[23] Yun Jong Pil, Choi SungHoo, Seo Boyeul, e Kim Sang Woo. Real-time vision-based defect inspection for high-speed steel products. *Optical Engineering*, 47(7):077204–1, 2008. 10158131 real-time vision-based defect inspection high-speed steel bar-in-coil steel-making industry product quality product quantity nonuniform brightness distribution image noise cylindrical shape. URL: http://dx.doi.org/10.1117/1.2957958, doi:10.1117/1.2957958.

[24] Xue-Wu Zhang, Yan-Qiong Ding, Yan-Yun Lv, Ai-Ye Shi, e Rui-Yu Liang. A vision inspection system for the surface defects of strongly reflected metal based on multi-class svm. *Expert Systems with Applications*, 38(5):5930–5939, 2011. Compilation and indexing terms, Copyright 2016 Elsevier Inc. 20110513630508 Automatic inspection system Classification results Defect classification Defects detection Feature extraction and classification Image preprocessing Image processing - methods Kernel function Metal surface defects Multiclass SVM Parameters setting Pattern recognition algorithms Smoothing methods Spectral measure Strongly reflected metal Testing process Vision inspection systems Vision systems. URL: http://dx.doi.org/10.1016/j.eswa.2010.11.030, doi:10.1016/j.eswa.2010.11.030.